UNIVERSITY OF COPENHAGEN
DEPARTMENT OF COMPUTER SCIENCE



Ph.D. Thesis

Vésteinn Snæbjarnarson

Transducing Language Models

Advisors: Serge Belongie and Ryan Cotterell
Assessment committee: David Chiang, Mirella Lapata, and Anders Søgaard

This thesis has been submitted to the PhD School of The Faculty of Science,
University of Copenhagen on August 31, 2026

Errata and updates are maintained at
https://vesteinn.is/tlm/

Abstract

Language models assign probabilities to strings of symbols, but the distribution an application needs can be a different one. A model over byte-pair tokens does not directly expose word-level probabilities. A model over DNA sequences does not directly score sequences of amino acids. A model over cased text scores a lowercased string only as one casing among many. Post-processing the model's outputs can produce samples in the desired representation, but it does not by itself provide a probabilistic language-model interface over that representation. Such an interface would support scoring target strings, conditioning on target prefixes, and returning a next-symbol distribution over the target alphabet. We call this gap the *string-mismatch problem*.

This thesis addresses the string-mismatch problem by introducing *transduced language models* (TLMs). A string drawn from one distribution and pushed through a deterministic map is a draw from a new distribution. When that map is encoded by a finite-state transducer, we show how to compose the transducer with a pretrained language model. The resulting model has a full autoregressive interface over the transducer's target alphabet without retraining. We express a target-prefix probability as the source-model mass over the set of all source strings that map to that prefix or its extensions. We then show how this set can be decomposed so that its mass can be calculated more efficiently, and give sufficient conditions for a finite decomposition.

Exact enumeration computes the prefix probability by summing over the decomposition, but the number of terms may be prohibitively large. Dropping low-probability source prefixes from the enumeration reduces time and memory use but underestimates the TLM prefix probability. We remove this bias with a sequential Monte Carlo estimator based on reweighted sampling without replacement. The estimator terminates with probability one even when the decomposition has an infinite number of terms. In our experiments, it has lower variance and is more efficient than with-replacement SMC baselines.

The final research chapter applies TLMs to psycholinguistics, where surprisal theory predicts greater processing difficulty for linguistic units that are less predictable in context. The method lets us keep the pretrained model fixed and compute surprisal for units suited to the scientific question rather than only for the model's symbols. We demonstrate the approach on eye-tracking data with the model's own symbols, characters, and several notions of a word as units of analysis.

Resumé

Sprogmodeller tildeler sandsynligheder til strenge af symboler, men en given anvendelse kan kræve en sandsynlighedsfordeling over en anden repræsentation. En model over byte-pair-tokens giver ikke direkte adgang til sandsynligheder på ordniveau. En model over DNA-sekvenser kan ikke direkte tildele sandsynligheder til aminosyresekvenser. En model, der skelner mellem store og små bogstaver, scorer en streng med små bogstaver som blot én af mange mulige skrivemåder. Efterbehandling af modellens output kan generere strenge i den ønskede repræsentation, men giver ikke i sig selv en probabilistisk sprogmodelgrænseflade for denne repræsentation. En sådan grænseflade ville gøre det muligt at score outputstrenge, betinge modellens fordeling på outputpræfikser og returnere en sandsynlighedsfordeling over det næste symbol i outputalfabetet. Vi kalder denne forskel for *string-mismatch-problemet*.

Denne afhandling behandler string-mismatch-problemet ved at introducere *transducerede sprogmodeller* (TLM'er). Hvis en streng trækkes fra én fordeling og føres gennem en deterministisk afbildning, følger resultatet en ny fordeling. Når afbildningen repræsenteres af en transducer med et endeligt antal tilstande, viser vi, hvordan transduceren kan sammensættes med en prætrænet sprogmodel. Den resulterende model har en fuld autoregressiv grænseflade over transducerens outputalfabet uden gentræning. Vi udtrykker sandsynligheden for et outputpræfiks som inputmodellens samlede sandsynlighedsmasse over alle inputstrenge, der afbildes til præfikset eller en forlængelse af det. Dernæst viser vi, hvordan denne mængde kan opdeles, så dens masse kan beregnes mere effektivt, og angiver tilstrækkelige betingelser for, at opdelingen har endeligt mange led.

Eksakt opregning beregner præfikssandsynligheden ved at summere opdelingens led, men antallet af led kan være uoverkommeligt stort. At udelade inputpræfikser med lav sandsynlighed fra opregningen reducerer tids- og hukommelsesforbruget, men undervurderer TLM'ens præfikssandsynlighed. Vi fjerner dette bias med en sekventiel Monte Carlo-estimator baseret på stikprøveudtagning uden tilbagelægning og omvægtning. Estimatoren terminerer med sandsynlighed én, selv når opdelingen har uendeligt mange led. I vores eksperimenter har den lavere varians og er mere effektiv end SMC-referencemetoder, der benytter stikprøveudtagning med tilbagelægning.

Afhandlingens sidste forskningskapitel anvender TLM'er i psykolingvistikken, hvor surprisal-teorien forudsiger større bearbejdningsvanskeligheder for sproglige enheder, der er mindre forudsigelige i konteksten. Metoden lader os holde den prætrænede model uændret og beregne surprisal for enheder valgt ud fra det videnskabelige spørgsmål frem for kun for modellens symboler. På øjenbevægelsesdata demonstrerer vi tilgangen med analyseenheder, der omfatter modellens egne symboler, tegn og ord afgrænset på flere forskellige måder.

Acknowledgments

At the start of 2016, while working as a software engineer, I made a New Year's resolution to understand how neural networks work. As is often the case with such resolutions, the year flew by, and my understanding had not deepened all that much. More than a decade on from the failed resolution, I consider myself fortunate to be able to think about machine learning for a living, with the freedom academia affords. The route was neither direct nor carefully planned. It took a fair bit of work and persistence, a good amount of luck, and some tactical improvisation. But none of it would have been possible without the support of many people along the way.

First, I would like to thank my advisors, Serge Belongie and Ryan Cotterell: Serge for taking me on as a PhD student and supporting me as I made the move into computer vision, and Ryan for assuming a larger role as an advisor during the latter half of my studies, as my research shifted toward formal language theory and algorithms. I also offer my sincerest thanks to Tim Vieira, who served in many ways as an additional advisor for the work presented here.

I also extend my thanks to the assessment committee—David Chiang, Mirella Lapata, and Anders Søgaard—for taking the time to examine this thesis. I am also immensely grateful to the staff at the Pioneer Centre for AI in Copenhagen and the ELLIS program for supporting my fellowship and facilitating my year-long visit to ETH Zürich. I especially thank Michelle Cumming Løkkegaard, Björg Birkholm Magnúsdóttir, and Patrizia Napoli, who ensured that I never had to worry about logistics during this time. I also thank the Danish National Research Foundation (DNRF grant P1) for funding this research.

Over the years, I was advised and supported by a great group of postdocs: Sagie Benaim, Stella Frank, Nico Lang, Ioannis Siglidis, and Josef Valvoda in Copenhagen, and Reda Boumasmoud, Brian DuSell, and Tim Vieira in Zürich. I am also grateful for the collaborations and the time I spent with my fellow researchers at the Pioneer Centre for AI: Zhaochong An, Peter Ebert Christensen, Jiaang Li, Wenyan Li, Sebastian Loeschcke, Noah Rothenberger, Srishti Yadav, Julia Machnio, Desmond Elliott, Mikkel Fruelund Odgaard, Maria Elkjær Montgomery, Mads Nielsen, Mostafa Mehdipour Ghazi, Silpa Vadakkeeveetil Sreelatha, Nadav Borenstein, Maxim Khomiakov, Mads Toftrup, Kiril Vadimovic Klein, Jakob Ambsdorf, Dan Wang, Asbjørn Munk, Anshuk Uppal, Christian Igel, Mads Herbert Kerrn, and Frederik Warburg. Similarly, at ETH Zürich, I thank Afra Amini, Alexandra Butoi, Kevin Du, Yahya Emara, Juan Luis Gastaldi, Mario Giulianelli, Alexander Hoyle, Jiaoda Li, Tiānyǔ Liú, Yufei Liu, Franz Nowak, Andreas Opedal, Clemente Pasti, Manuel de Prada Corral, Shauli Ravfogel, Francesco Ignazio

Re, Karolina Stańczak, Niklas Stoehr, Anej Svete, Eleftheria Tsipidi, Jannek Ulm, and Alex Warstadt. I am particularly grateful to Samuel Kiegeland for our collaboration on the three articles that form the basis of this thesis. I also thank Thomas Hikaru Clark, Jacob Hoover Vigly, Zach Hopton, and Ben LeBrun for their feedback on the psycholinguistic work.

I thank Hafsteinn Einarsson and Kristján Jónasson for hosting me during an extended visit at the University of Iceland. I am also very grateful to all my other co-authors and office friends across these three institutions.

I also thank those who supported me in my earlier career. In particular, I thank Haukur Barri Símonarson and Vilhjálmur Þorsteinsson at Miðeind, where I got the chance to work on Icelandic NLP and train my first neural networks. I thank Arnar Jensson, who introduced me to NLP in the first place, albeit in a different era. I am grateful to Ólafur S. Andrésson for introducing me to bioinformatics and research in general.

Finally, this thesis would not have been possible without the incredible support of my family: my parents Þórdís and Snæbjörn; my daughters Lísa and Eva; my in-laws Gill and Ian; my brother Jökull and his partner María; sister-in-law Vicki, Mark, and Joe; Lísa's other parents Alba and Tóti; and my grandparents Gísli and Reisí. Most of all, I thank my wife Helen, who supported me in every way possible and held the fort on so many occasions. ♥

Table of Contents

Abstract . . . iii
Resumé . . . iv
Acknowledgments . . . v
List of Figures . . . xi
List of Tables . . . xii
List of Algorithms . . . xiii
List of Formal Results . . . xiv
List of Examples . . . xv
List of Publications . . . xvi
Contribution Statement . . . xix
Note on Prior Publication . . . xx
Notation . . . xxii
Conventions . . . xxii
Glossary . . . xxiii

1 Introduction **1**
1.1 The string-mismatch problem . . . 1
1.2 Prior work . . . 4
1.2.1 Problem-specific approaches . . . 4
1.2.2 Model transformation and inference . . . 6
1.3 Research questions . . . 8
1.4 The TLM approach . . . 8
1.5 Contributions and roadmap . . . 9

2 Background **11**
2.1 Strings, alphabets, and languages . . . 11
2.2 Language models . . . 11
2.3 Cylindrical sets . . . 13
2.4 Finite-state transducers . . . 14
2.4.1 Transducer variants . . . 15
2.4.2 Operations . . . 16

2.4.3 Limitations of transducers . . . 17

3 Transduced language models **18**
3.1 The pushforward construction . . . 18
3.2 Decomposing the precover . . . 19
3.3 Conditions for finite decompositions . . . 22
3.4 Example decompositions . . . 29
3.5 Relation to prior constructions . . . 31
3.5.1 The tokens-to-characters special case . . . 32

4 Inference **34**
4.1 The autoregressive interface . . . 34
4.2 Exact inference . . . 35
4.2.1 Getting started: instantiating the checks with the precover machine . . . 39
4.3 Efficient algorithms . . . 42
4.3.1 Frontier computation . . . 42
4.3.2 Frontier-based checks . . . 46
4.3.3 IP-universality shortcut . . . 47
4.3.4 Pruning . . . 49
4.3.5 Faster next-symbol decomposition . . . 51
4.3.6 Next-symbol decomposition shortcuts . . . 55
4.3.7 Memoization and batching of $p_{\mathcal{X}}$ . . . 55
4.3.8 Realizing the autoregressive interface . . . 56
4.3.9 All-universal fast path . . . 57
4.3.10 Label pushing . . . 58
4.4 Experiments . . . 58

5 Unbiased inference via stochastic estimation **62**
5.1 Unbiased estimation of transduced language models . . . 63
5.1.1 Beam summing over the precover decomposition . . . 64
5.1.2 Stochastic pruning . . . 68
5.1.3 Unbiasedness guarantees . . . 70
5.1.4 Adaptive-budget SWOR and almost-sure halting . . . 74
5.1.5 SMC with multinomial resampling . . . 77
5.1.6 Relation to prior work . . . 78
5.2 Experiments . . . 78
5.2.1 Byte-to-word transduction on real text . . . 79
5.2.2 Comparing compute–variance frontiers . . . 80
5.2.3 Transducing DNA language models . . . 81
5.2.4 Variance across estimators . . . 84

5.3 Summary . . . 84

6 Surprisal in psycholinguistics 85
6.1 Units . . . 87
6.1.1 Lost in whitespace . . . 89
6.1.2 Regular unit inventories . . . 91
6.1.3 A finite-alphabet representation of the unit parser . . . 92
6.2 Regions of interest . . . 93
6.3 From surprisal to reading times . . . 94
6.3.1 Baseline predictors . . . 95
6.3.2 Predictive power . . . 96
6.4 Experiments . . . 97
6.4.1 Unit inventories . . . 97
6.4.2 (Re-)processing the MECO corpus . . . 100
6.4.3 Estimating surprisal . . . 101
6.4.4 Analysis . . . 101
6.4.5 Results . . . 101
6.5 Reanalysis with unbiased prefix-probability estimates . . . 102
6.6 Chapter limitations . . . 103

7 Summary and discussion 105
7.1 Overview . . . 105
7.2 Research questions . . . 106
7.3 Limitations . . . 107
7.4 Future directions . . . 108
7.5 Closing remark . . . 109

A Inference: algorithms and experiments 110
A.1 Additional algorithmic material . . . 110
A.1.1 Combined universality . . . 110
A.1.2 All-universal fast path . . . 112
A.1.3 Backtracking . . . 113
A.1.4 State-based decomposition . . . 113
A.2 Experimental setup . . . 115
A.2.1 Datasets . . . 116
A.2.2 Models . . . 117
A.2.3 Parameters . . . 118
A.2.4 GPU usage . . . 118
A.2.5 Details on transducers used in experiments . . . 118
A.3 Additional experimental results . . . 120
A.3.1 Comparison to Vieira et al. (2025a) . . . 120

A.3.2 Jensen–Shannon divergence . . . 121
A.3.3 Decomposition size . . . 126
A.3.4 Cross-entropy and throughput . . . 128
A.3.5 Benchmarking the computational shortcut . . . 130

B Stochastic estimation: algorithms, proofs, and results 133
B.1 Sequential Monte Carlo over the precover . . . 133
B.1.1 Sequential Monte Carlo . . . 133
B.1.2 Intermediate targets and the `smc_rb` algorithm . . . 135
B.1.3 Unbiasedness of `smc_rb` . . . 139
B.1.4 A common tail-roulette prune . . . 140
B.1.5 Proof for adaptive-budget SWOR . . . 142
B.2 Sensitivity to pruning parameters . . . 146
B.3 Exact ground truth for bigram models . . . 147
B.4 The genetic-code worst case . . . 148
B.5 Additional byte-to-word results . . . 149

C Surprisal: units, data, and regression details 153
C.1 Prefix-freeness of h . . . 153
C.2 Unit transformations used in the experiments . . . 154
C.2.1 Characters . . . 154
C.2.2 Acontextual words . . . 154
C.2.3 Contextual words . . . 155
C.2.4 Transducer sizes . . . 155
C.3 Dataset details . . . 156
C.3.1 Unit overlap . . . 157
C.3.2 Unit and fixation visualizations . . . 158
C.4 Experimental details . . . 158
C.4.1 Computing surprisal . . . 160
C.4.2 GPU usage & runtime . . . 162
C.5 GAMM specification . . . 163
C.6 Details of the MECO reanalysis . . . 164
C.7 Additional results . . . 165

Bibliography 167

List of Figures

1.1 GPT-2 probabilities for lowercase-matching token strings . . . 2

3.1 An FST with an exceptional input . . . 31

4.1 A token-to-character FST . . . 59
4.2 A comma-separator insertion FST . . . 61
4.3 JSD and throughput: token-to-byte and PTB . . . 61

5.1 One adaptive-budget SWOR prune . . . 76
5.2 All-child SWOR versus sampled-extension SMC . . . 77
5.3 Unbiased SMC versus beam summing on the bigram testbed . 80
5.4 GPT-2 Large on the PTB testbed . . . 81
5.5 The accuracy–walltime frontier on both testbeds . . . 82
5.6 Accuracy and runtime for DNA-to-amino-acid transduction . . 83

6.1 One string at three levels . . . 86
6.2 The unit parser and separator alphabet . . . 93
6.3 Per-observation improvement by unit inventory . . . 102

A.1 An FST converting DNA sequences to amino acids . . . 120
A.2 Quotient and remainder growth with position . . . 128
A.3 Per-repeat JSD versus converted IP-universal states . . . 132

B.1 A source string aligned to a target prefix . . . 137
B.2 Sensitivity to pruning parameters . . . 147
B.3 Per-position bigram validation (paragraphs 0–5) . . . 150
B.4 Per-position bigram validation (paragraphs 6–9) . . . 151

C.1 Units and fixations: model tokens . . . 158
C.2 Units and fixations: acontextual (leading) words . . . 158
C.3 Units and fixations: acontextual (trailing) words . . . 159
C.4 Units and fixations: contextual words . . . 159
C.5 Units and fixations: character-level units . . . 159
C.6 GAMM smooth-term F-statistics . . . 166

List of Tables

5.1 The TLM algorithm family 66
5.2 Variance of each estimator across settings 84

6.1 Within-setting GAMM fit on MECO 103

A.1 Protein accession numbers for the DNA experiments 117
A.2 Training parameters for DNA model 118
A.3 GPT-2 training and validation metrics 118
A.4 Transducer sizes in the inference experiments 119
A.5 Exact comparison to genlm-bytes 122
A.6 JSD and throughput, token-to-byte (reference beam K=32) . 123
A.8 JSD and throughput, token-to-byte 123
A.7 Comparison to the token-to-byte beam baseline 124
A.9 JSD and throughput, PTB tokenization 124
A.10 JSD and throughput, DNA (candidate cap 5000–20000) 125
A.11 JSD and throughput, DNA (candidate cap 25000–30000) . . . 126
A.12 Mean and maximum decomposition sizes by threshold 127
A.13 Cross-entropy and throughput, token-to-byte (GPT-2, Llama-1B) 129
A.14 Cross-entropy and throughput, token-to-byte (Llama-8B, Phi-4) 130
A.15 JSD after converting IP-universal states 131

B.1 Per-paragraph head-to-head under GPT-2 Large 152

C.1 Sizes of the transducers used in experiments 155
C.2 Pipeline observation counts per unit inventory 156
C.3 Unit overlap with whitespace stripped 157
C.4 Unit overlap with whitespace kept 157
C.5 Hyperparameters for surprisal estimation 161
C.6 Contextual surprisal scoring throughput 162
C.7 Observation counts entering the GAMM 165
C.8 GAMM results across reading-time measures 165

List of Algorithms

4.1 Interface to the transduced language model (`prefix_prob`, `prob`, `next_dist`) . . . 35
4.2 The decomposition algorithm (`decompose`) . . . 36
4.3 State-based checks (`is_member`, `is_live`, `is_cylinder`) . . . 41
4.4 Frontier-based state machine (`run`, `step`, `closure`) . . . 43
4.5 Frontier-based checks (`is_cylinder`, `is_member`, `is_live`) . . . 48
4.6 IP-universal states (`fast_univ_filter`) . . . 50
4.7 Probability-mass pruning (`prune_prob_mass`) . . . 51
4.8 Frontier helpers (`is_exact_member`, `reachable_inputs`, `reachable_outputs`) . . . 53
4.9 Fast next-symbol decomposition (`decompose_next`) . . . 54
4.10 Faster implementation of `next_dist` using `decompose_next` . . . 57

5.1 Beam summing (`beam_summing`) and deterministic pruning . . . 65
5.2 The SWOR reducers (`swor`, `swor_adaptive`, `pps`, `systematic`) . . . 69

A.1 The combined-universality check (`combined_universal`) . . . 112
A.2 The all-universal fast path (`all_universal_next_dist`) . . . 114
A.3 Backtracking (`backtrack`) . . . 115
A.4 State-based precover decomposition (`dfa_decomposition`) . . . 116

B.1 Sequential Monte Carlo for evidence estimation (`smc_simple`, `resample`) . . . 134
B.2 The `smc_rb` algorithm and ESS-gated multinomial resampling under a twist . . . 138
B.3 Tail-roulette decorator for a pruning function. . . . 141

List of Formal Results

3.1 Equivalent characterizations of prefix monotonicity 23
3.2 Strict-prefix monotonicity and finite quotients 24
3.1 Sufficient conditions for finite decompositions 26

4.1 Correctness of `decompose` 37
4.1 Frontier containment 45
4.2 Generalized frontier containment 46
4.3 IP-universality shortcut 48
4.4 Non-cylinder monotonicity 55
4.5 Cylinder uniqueness 55

5.1 A backward recursion for the optimal twist 67
5.2 Pruning preserves the unpruned pool weights in expectation . 70
5.1 Unbiasedness 72
5.1 Adaptive-budget SWOR 76

A.1 Combined universality 111

B.1 Unbiasedness of `smc_rb` 139

List of Examples

2.1 Prefix probabilities 12
2.2 Cylinders and the prefix-base 13
2.3 A parity transducer 15

3.1 The lowercasing transducer 19
3.2 A remainder in Newspeak 20
3.3 Monotonicity and the prefix decomposition 22
3.4 Three ways for a state to be safe 27
3.5 Merging consecutive backticks 29
3.6 An infinite remainder 29
3.7 A one-element remainder 30

4.1 Tracing `decompose` on the lowercasing transducer 37
4.2 Building the precover machine 39
4.3 Tracing the frontier computation 44
4.4 When the per-state shortcut fails 46

5.1 An infinite quotient 74

6.1 Segmentation ambiguity in Mandarin 88
6.2 Whitespace does not change the units 89
6.3 Computing a next-unit probability with a trailing SEP 92
6.4 Limits of delimiter segmentation 98
6.5 Segmenting commas and colons by context 99

B.1 Beam summing versus SMC on a uniform block code 148

List of Publications

This thesis is based on three articles. Their material has been edited and reorganized to unify notation and integrate the articles into a single narrative. The *Note on Prior Publication* (page xx) describes these changes.

1. Vésteinn Snæbjarnarson*, Samuel Kiegeland*, Tianyu Liu, Reda Boumasmoud, Ryan Cotterell, and Tim Vieira (2026a). "Transducing Language Models". In: *The Fourteenth International Conference on Learning Representations*. ICLR 2026. URL: https://openreview.net/forum?id=qOyF214xmg
2. Vésteinn Snæbjarnarson, Samuel Kiegeland, Manuel de Prada Corral, Ryan Cotterell, and Tim Vieira (2026b). "Stochastic Estimation of Transduced Language Models". In: *arXiv preprint arXiv:2608.27428*. URL: https://arxiv.org/abs/2608.27428
3. Samuel Kiegeland, Vésteinn Snæbjarnarson, Tim Vieira, and Ryan Cotterell (2026). "On the Proper Treatment of Units in Surprisal Theory". In: *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2026. URL: https://aclanthology.org/2026.acl-long.1485/

The publications below were co-authored during the PhD but are not included in this thesis.

1. Vésteinn Snæbjarnarson, Anej Svete, Josef Valvoda, Reda Boumasmoud, Brian DuSell, and Ryan Cotterell (2026c). "Causally Evaluating the Learnability of Formal Language Tasks". In: *Forty-third International Conference on Machine Learning*. ICML 2026. URL: https://openreview.net/forum?id=sYUCXUeApn
2. Annika Simonsen, Barbara Scalvini, Uni Johannesen, Iben Nyholm Debess, Hafsteinn Einarsson, and Vésteinn Snæbjarnarson (2026). "BRAGD: Constrained Multi-Label POS Tagging for Faroese". In: *Proceedings of the 15th Language Resources and Evaluation Conference (LREC 2026)*. LREC 2026. URL: https://aclanthology.org/2026.lrec-1.913/
3. Jannek Ulm, Kevin Du, and Vésteinn Snæbjarnarson (2025). "Contrastive

Decoding for Synthetic Data Generation in Low-Resource Language Modeling". In: *Proceedings of the First BabyLM Workshop*. BabyLM @ EMNLP 2025. URL: https://aclanthology.org/2025.babylm-main.2/

4. Vésteinn Snæbjarnarson, Kevin Du, Niklas Stoehr, Serge Belongie, Ryan Cotterell, Nico Lang, and Stella Frank (2025). "Taxonomy-Aware Evaluation of Vision-Language Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR 2025. URL: https://openaccess.thecvf.com/content/CVPR2025/html/Snaebjarnarson_Taxonomy-Aware_Evaluation_of_Vision-Language_Models_CVPR_2025_paper.html
5. Shauli Ravfogel, Anej Svete, Vésteinn Snæbjarnarson, and Ryan Cotterell (2025). "Gumbel Counterfactual Generation From Language Models". In: *The Thirteenth International Conference on Learning Representations*. ICLR 2025. URL: https://openreview.net/forum?id=TUC0ZT2zIQ
6. Sebastian Loeschcke*, Mads Toftrup*, Michael J. Kastoryano, Serge Belongie, and Vésteinn Snæbjarnarson (2024). "LoQT: Low-Rank Adapters for Quantized Pretraining". In: *Advances in Neural Information Processing Systems*. Vol. 37. NeurIPS 2024. URL: https://proceedings.neurips.cc/paper_files/paper/2024/hash/d0c6bc641a56bebee9d985b937307367-Abstract-Conference.html
7. Niklas Stoehr, Kevin Du, Vésteinn Snæbjarnarson, Robert West, Ryan Cotterell, and Aaron Schein (2024). "Activation Scaling for Steering and Interpreting Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Findings of EMNLP 2024. URL: https://aclanthology.org/2024.findings-emnlp.479/
8. Evgenia Ilyinskaya, Vésteinn Snæbjarnarson, Hanne Krage Carlsen, and Björn Oddsson (2024). "Brief communication: Small-scale geohazards cause significant and highly variable impacts on emotions". In: *Natural Hazards and Earth System Sciences* 24.9. NHESS 2024. URL: https://nhess.copernicus.org/articles/24/3115/2024/
9. Kevin Du, Vésteinn Snæbjarnarson, Niklas Stoehr, Jennifer White, Aaron Schein, and Ryan Cotterell (2024). "Context versus Prior Knowledge in Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2024. URL: https://aclanthology.org/2024.acl-long.714/
10. Steinunn Rut Friðriksdóttir, Annika Simonsen, Atli Snær Ásmundsson, Guðrún Lilja Friðjónsdóttir, Anton Karl Ingason, Vésteinn Snæbjarnarson, and Hafsteinn Einarsson (2024). "Ice and Fire: Dataset on Sentiment, Emotions, Toxicity, Sarcasm, Hate speech, Sympathy and More in Icelandic Blog Comments". In: *Proceedings of the Fourth Workshop on Threat, Aggression & Cyberbullying @ LREC-COLING-2024*. TRAC @ LREC-

COLING 2024. URL: https://aclanthology.org/2024.trac-1.9/

11. Nico Lang, Vésteinn Snæbjarnarson, Elijah Cole, Oisin Mac Aodha, Christian Igel, and Serge Belongie (2024). "From Coarse to Fine-Grained Open-Set Recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR 2024. URL: https://openaccess.thecvf.com/content/CVPR2024/html/Lang_From_Coarse_to_Fine-Grained_Open-Set_Recognition_CVPR_2024_paper.html
12. Peter Ebert Christensen, Vésteinn Snæbjarnarson, Andrea Dittadi, Serge Belongie, and Sagie Benaim (2024). "Assessing Neural Network Robustness via Adversarial Pivotal Tuning". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. WACV 2024 (oral). URL: https://openaccess.thecvf.com/content/WACV2024/html/Christensen_Assessing_Neural_Network_Robustness_via_Adversarial_Pivotal_Tuning_WACV_2024_paper.html
13. Idan Schwartz*, Vésteinn Snæbjarnarson*, Hila Chefer, Serge Belongie, Lior Wolf, and Sagie Benaim (2023). "Discriminative Class Tokens for Text-to-Image Diffusion Models". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. ICCV 2023. URL: https://openaccess.thecvf.com/content/ICCV2023/html/Schwartz_Discriminative_Class_Tokens_for_Text-to-Image_Diffusion_Models_ICCV_2023_paper.html
14. Svanhvít Lilja Ingólfsdóttir, Pétur Ragnarsson, Haukur Jónsson, Haukur Símonarson, Vilhjálmur Þorsteinsson, and Vésteinn Snæbjarnarson (2023). "Byte-Level Grammatical Error Correction Using Synthetic and Curated Corpora". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2023 (oral). URL: https://aclanthology.org/2023.acl-long.402/
15. Vésteinn Snæbjarnarson, Annika Simonsen, Goran Glavaš, and Ivan Vulić (2023). "Transfer to a Low-Resource Language via Close Relatives: The Case Study on Faroese". In: *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*. NoDaLiDa 2023. URL: https://aclanthology.org/2023.nodalida-1.74/

Contribution Statement

I had a key role in all three articles included in this thesis. They were, however, highly collaborative, and my role varied among them. The following statement makes those roles explicit.

Transducing Language Models. The project was conceptualized in collaboration with Ryan Cotterell and Tim Vieira. The algorithms were developed by me in collaboration with Samuel Kiegeland, Tim Vieira, and Ryan Cotterell. Software development was done by me in collaboration with Samuel Kiegeland, Tim Vieira, and Tianyu Liu. The experiments were run by me and Samuel Kiegeland. Theoretical statements and proofs were written and checked in collaboration with Samuel Kiegeland, Tim Vieira, Ryan Cotterell, and Reda Boumasmoud. I wrote the article in collaboration with Samuel Kiegeland, Reda Boumasmoud, Ryan Cotterell, and Tim Vieira.

Stochastic Estimation of Transduced Language Models. The project was conceptualized in collaboration with Tim Vieira, Ryan Cotterell, and Samuel Kiegeland. The algorithms and theory were developed by me in collaboration with Tim Vieira, Ryan Cotterell, and Manuel de Prada Corral. Software development was done by me. All experiments were run by me. The article was written by me, Samuel Kiegeland and Tim Vieira.

On the Proper Treatment of Units in Surprisal Theory. The paper was conceptualized in collaboration with Samuel Kiegeland and Ryan Cotterell. My contribution focused on the mathematical framing and the unit and region-of-interest separation. All data processing and original experiments were run by Samuel Kiegeland. For the thesis, I ran the SMC-based reanalysis reported in Section 6.5. Writing was done in collaboration with Samuel Kiegeland, Ryan Cotterell, and Tim Vieira.

Note on Prior Publication

This thesis incorporates material from three papers co-authored during my PhD studies. The body chapters integrate, rearrange, edit, and expand material from these papers into a single narrative. Chapter-level and selected section-level attributions identify the papers on which each part is based.

- **Chapter 1 – Introduction.** The lowercasing, tokenization, byte-encoding, and DNA-to-amino-acid examples in §1.1 are reproduced or adapted from Snæbjarnarson et al. (2026a), as is some of the surrounding framing. The prior-work survey in §1.2 synthesizes and extends related-work discussions from all three articles. The research questions, overview of the TLM approach, and roadmap were written for the thesis.
- **Chapter 2 – Background.** Collects the shared preliminaries that the body chapters rely on, reproduced or adapted from Snæbjarnarson et al. (2026a) and Kiegeland et al. (2026).
- **Chapter 3 – Transduced language models.** Reproduces and adapts material from Snæbjarnarson et al. (2026a).
- **Chapter 4 – Inference.** Reproduces and adapts material from Snæbjarnarson et al. (2026a).
- **Chapter 5 – Unbiased inference via stochastic estimation.** Reproduces and adapts material from Snæbjarnarson et al. (2026b).
- **Chapter 6 – Surprisal in psycholinguistics.** Reproduces and adapts material from Kiegeland et al. (2026) and, for the SMC-based reanalysis (§6.5), from Snæbjarnarson et al. (2026b).
- **Chapter 7 – Summary and discussion.** The discussion of unit choice adapts the conclusion of Kiegeland et al. (2026), and the empirical-scope limitation adapts Snæbjarnarson et al. (2026a).

Editorial changes. The papers' notation was unified according to the conventions introduced in the front-matter *Notation* pages. The papers' standalone introductions and background material were consolidated into the thesis-wide Chapters 1 and 2 and the chapter openings. References to "this paper" were replaced with thesis-internal cross-references.

Supplementary material from the papers was either moved to the thesis appendices or incorporated into the body chapters. Appendix A reproduces and adapts supplementary material from Snæbjarnarson et al. (2026a), Appendix B from Snæbjarnarson et al. (2026b), and Appendix C from Kiegeland et al. (2026) and, for the MECO reanalysis details, from Snæbjarnarson et al. (2026b). Material incorporated into the body includes the transducer background, finiteness proofs, example decompositions, and efficient inference algorithms from Snæbjarnarson et al. (2026a), together with three proofs deferred to the appendix of Snæbjarnarson et al. (2026b). Further editorial changes improve narrative flow and clarify the scope of algorithmic and experimental claims.

Several worked examples are new to the thesis: the three examples of Chapter 2 (Examples 2.1 to 2.3), the comparison of three examples on the monotonicity hierarchy (Example 3.3), the three algorithm traces of Chapter 4 (Examples 4.1 to 4.3), and the unit-scoring example of Chapter 6 (Example 6.3). The remaining examples reproduce or adapt material from the papers, sometimes restructuring figures and captions.

No quantitative results have been altered from the underlying papers. Some descriptions have been qualified for precision without changing the underlying experimental findings. The proofs of propositions 3.1 and 3.2 and Lemma 3.1 have been expanded and corrected where the conference version compressed or omitted intermediate arguments, without changing the statements or conclusions of those results. In Chapter 4, the finiteness hypothesis and the proof of Theorem 4.1 were revised to cover the recursive implementation, and the conclusion of Proposition 4.3 was corrected to maximal-quotient membership. The direction labels in the proof of Proposition 4.1 were corrected to match the two arguments, without changing the identity proved.

The per-chapter attributions above also cover figures reproduced or adapted from the three articles.

Use of generative AI. Generative AI was used as a writing, editing, and proofreading aid in preparing this thesis. In particular, it assisted with integrating the papers into a single narrative, unifying notation and cross-references, and editing for style, consistency, and flow. The author takes full responsibility for the thesis, including all text edited with AI assistance.

Notation

This chapter collects the notation used throughout the thesis, allowing each body chapter to rely on a common set of conventions.

Conventions

Color encodes domain. Magenta denotes source-domain objects ($\boldsymbol{x}$, $\mathcal{X}$, $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$), while green denotes target-domain objects ($\boldsymbol{y}$, $\mathcal{Y}$, f). In Ch 6, green also denotes units and unit-level constructs ($\boldsymbol{u}$, $\mathcal{U}$). In Ch 6, blue denotes the separator encoding (Ξ, $\boldsymbol{\xi}$, h). Black denotes domain-independent objects, such as transducer states and transitions.

Font encodes type. Lowercase italic denotes symbols (x, y), while bold italic denotes strings ($\boldsymbol{x}$, $\boldsymbol{y}$, $\boldsymbol{u}$). Calligraphic type generally denotes alphabets and sets ($\mathcal{X}$, $\mathcal{Y}$, $\mathcal{U}$, $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$). Sans-serif capitals denote the structural components of a transducer (S, T, I, F). Typewriter type denotes transducers (`f`), pseudocode identifiers (`decompose`, `Q`), and concrete strings in examples (`abc`, `xyz`).

Prefix probabilities. An arrow above a probability symbol marks a prefix probability. For example, $\overrightarrow{p_{\mathcal{X}}}$ denotes the prefix probability induced by $p_{\mathcal{X}}$; $\overrightarrow{p_{\mathcal{Y}}}$ and $\overrightarrow{p_{\mathcal{U}}}$ are defined analogously.

Glossary

Alphabets and strings

Notation	Gloss
ε	The empty string.
EOS	End-of-string symbol.
$x, x' \in \mathcal{X}$	Source-alphabet symbols.
$y, y' \in \mathcal{Y}$	Target-alphabet symbols.
$u \in \mathcal{U}$	A unit (Ch 6).
SEP	Distinguished unit separator; SEP $\notin \Xi$.
$\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}^*$	Source strings.
$\boldsymbol{y}, \boldsymbol{y}' \in \mathcal{Y}^*$	Target strings.
$\boldsymbol{x}\cdot\boldsymbol{x}'$ or $\boldsymbol{x}\boldsymbol{x}'$	String concatenation.
$\boldsymbol{x} \preceq \boldsymbol{x}'$	Prefix relation; $\prec$ for strict prefix.
$Z, Z' \subseteq \mathcal{X}^*$	Sets of source strings.
$\langle Z \rangle$	Cylinder set spanned by Z: $\langle Z \rangle \stackrel{\text{def}}{=} Z\mathcal{X}^*$, all extensions of strings in Z (Section 2.3).
$\text{pf}(Z)$	Prefix-base of Z: the prefix-free subset of Z consisting of strings with no proper prefix in Z.
Σ^*, Σ^+	Kleene closure (with / without ε).

Language models and prefix probabilities

Notation	Gloss
$p_{\mathcal{X}}$	Language model over source strings.
$p_{\mathcal{Y}}$	Language model over target strings.
$p_{\mathcal{U}}$	Unit-level language model (Ch 6).
p_{H}	Implicit human language model (Ch 6).
$X \sim p_{\mathcal{X}}$	$\mathcal{X}^*$-valued random variable.
$f(X) \sim p_{\mathcal{Y}}$	Pushforward under f.
$\overrightarrow{p_{\mathcal{X}}}$	Prefix probability.

Maps and finite-state transducers

Notation	Gloss
$f\colon \mathcal{X}^* \to \mathcal{Y}^*$	String-to-string function.
$f^{-1}(\boldsymbol{y})$	Preimage of $\boldsymbol{y}$ under f.
$\mathtt{f}$	Transducer implementation of f.
$\hookrightarrow \mathcal{X}, \hookrightarrow \mathcal{Y}$	Input and output alphabets of $\mathtt{f}$.
$\hookrightarrow \mathsf{S}$	State set.
$\hookrightarrow \mathsf{T}$	Transition set.
$\hookrightarrow \mathsf{I} \subseteq \mathsf{S}$	Initial states.
$\hookrightarrow \mathsf{F} \subseteq \mathsf{S}$	Accepting states.
$\hookrightarrow \mathsf{U} \subseteq \mathsf{S}$	IP-universal states.
$s, s' \in \mathsf{S}$	States.
$(s \xrightarrow{x:y} s') \in \mathsf{T}$	A transition scanning x and emitting y.
$\mathtt{f} \circ \mathtt{f}'$	Transducer composition.
$\mathsf{proj}_{\mathcal{X}}(\mathtt{f})$	Input projection of $\mathtt{f}$.
$\mathtt{f}_{[s]}$	Force-start $\mathtt{f}$ in state s.
$[\![\mathtt{f}]\!]$	The relation defined by $\mathtt{f}$.
$\mathsf{trim}(\mathtt{f})$	Trimming: drop states/arcs on no accepting path.
$\mathsf{determinize}(\mathtt{f})$	Input determinization.
$\boldsymbol{y}\mathcal{Y}^*$	Copy-transducer for the cylinder $\boldsymbol{y}\mathcal{Y}^*$ (or the corresponding relation).

Precover, quotient, and remainder (Ch 3)

Notation	Gloss
$\mathcal{P}(\boldsymbol{y})$	Precover of $\boldsymbol{y}$: source strings whose image extends $\boldsymbol{y}$.
$\mathcal{C}(\boldsymbol{y})$	Largest cylinder set contained in $\mathcal{P}(\boldsymbol{y})$.
$\mathcal{Q}(\boldsymbol{y})$	Quotient of $\mathcal{P}(\boldsymbol{y})$.
$\mathcal{R}(\boldsymbol{y})$	Remainder of $\mathcal{P}(\boldsymbol{y})$.
$\mathcal{F}$	Frontier of $(s, \boldsymbol{b})$ pairs reachable while reading a source string.
$\boldsymbol{b}$	Buffer of pending output symbols.

Decomposition algorithms (Ch 4)

Notation	Gloss
`decompose`$(\boldsymbol{y})$	Prefix-decomposition algorithm; returns (`Q`, `R`).
`decompose_next`$(\boldsymbol{y})$	Joint decomposition of all one-symbol extensions $\boldsymbol{y}y$.
`Q`, `R`	Computed quotient and remainder sets.
$\mathscr{P}$	Queue of pending source prefixes
`is_cylinder`$(\boldsymbol{x}, \boldsymbol{y})$	Cylinder check: every extension covers, $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$.
`is_member`$(\boldsymbol{x}, \boldsymbol{y})$	Membership check: $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$.
`is_live`$(\boldsymbol{x}, \boldsymbol{y})$	Liveness check: some extension covers, $\exists\, \boldsymbol{x}' \colon \boldsymbol{x}\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y})$.
`fast_univ_filter`	IP-universality shortcut for the cylinder check.
$\mathtt{P}_{\boldsymbol{y}}$	Precover machine: determinized, trimmed DFA accepting $\mathcal{P}(\boldsymbol{y})$.
$\mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$	Frontier reached by scanning the source string $\boldsymbol{x}$.
$\mathtt{step}_{\boldsymbol{y}}$	Advance the frontier by one source symbol.
$\mathtt{closure}_{\boldsymbol{y}}$	The ε-closure of the frontier.
S	Powerstate: set of states reachable on the same input.
τ	Probability-mass pruning threshold.
n_{max}	Cap on the candidate-set size during pruning.
K	Beam size of the genlm-bytes baseline.

Stochastic estimation (Ch 5)

Notation	Gloss
M	Fixed or maximum particle count.
$\boldsymbol{x}^{(m)}$, $w^{(m)}$	The m-th particle's source prefix and importance weight.
$\mathscr{P}$	The particle pool.
q, $\overrightarrow{q}$	Proposal distribution and its conditional prefix probabilities.
ϕ	Potential weighting a source string.
Z, $\hat{Z}$	Evidence (the estimand) and its estimate. $\hat{Z}_t$ is the estimate at target position t.
ψ, ψ^*	A twist is a nonnegative weighting on source prefixes. The optimal twist is the conditional probability that a source completion covers the target.
μ_t	Intermediate target distribution at target position t.
$\mathcal{A}(\boldsymbol{x})$	The live next source symbols after $\boldsymbol{x}$.
$\lambda_t(\boldsymbol{x})$	Twisted mass of the live next source symbols after $\boldsymbol{x}$ at target position t.
$\nu_t(\boldsymbol{x})$	The position-t proposal normalizer, a member's stop mass plus its twist-weighted live-continuation mass.
A	Current-position accumulated total, returned as $\hat{Z}_t$.
ϵ	Halting constant in the effective running total $A_{\epsilon,t}$.
ESS, η	Effective sample size and the resample trigger: resample when ESS $< \eta M$.
ρ	Scales the adaptive-budget prune's per-particle threshold and retained count.
$\mathcal{S}$	Child set sampled without replacement: size M under fixed-budget SWOR and size m under adaptive-budget SWOR.
$w^{(i)}/\pi_i$	Horvitz–Thompson reweighting of a surviving particle.

Units and regions of interest (Ch 6)

Notation	Gloss
$\mathcal{U}$	Unit inventory chosen by the modeler.
Ξ	Alphabet over which units are strings.
u_t	The t-th unit in an utterance.
$\boldsymbol{u} = u_1 \cdots u_T$	An utterance: a sequence of units.
$\boldsymbol{u}_{[i,j)}$	Region of interest: subspan $u_i \cdots u_{j-1}$.
ρ	Unit parser (stochastic, taken deterministic here).
ρ^{-1}	Realization relation, the inverse of ρ.
$h(u)$	Unit-to-string homomorphism appending the separator: $h(u) = \boldsymbol{\xi}_u \text{SEP}$.
D	Delimiter set of the acontextual segmentations.
r	Reading-time measurement.
Δ_{llh}	Mean per-observation improvement in held-out log-likelihood.

Named models and transducers (Chs 4–6)

Notation	Gloss
p_{gpt2}	GPT-2 Large.
p_{llama1B}	LLaMA 3.2-1B.
p_{llama8B}	LLaMA 3.1-8B.
p_{phi4}	Phi-4.
p_{dna}	DNA model trained on the human genome.
p_{bigram}	Bigram model.
f_{α}	Token-to-byte transducer.
f_{ptb}	Penn Treebank segmentation transducer.
$\mathsf{f}_{\text{dna2aa}}$	Transducer from nucleobases to amino-acid symbols.
$\mathsf{f}_{\text{L}}, \mathsf{f}_{\text{T}}$	Acontextual delimiter transducers (leading / trailing attribution).

Chapter 1

Introduction

This chapter contains material reproduced and adapted from Snæbjarnarson et al. (2026a), with editorial changes for narrative consistency. The prior-work survey also adapts material from Snæbjarnarson et al. (2026b) and Kiegeland et al. (2026).

1.1 The string-mismatch problem

A language model defines a probability distribution over strings.[1] And the symbols those strings are built from, the model's *alphabet*, are a modeling choice fixed at training time. That distribution often doesn't match the one a downstream application needs, whether because the alphabets differ or because the probability mass is grouped in a way that does not match the question being asked. For a natural-language model trained on byte-pair-encoded subwords, that distribution is over subword sequences rather than words. For a DNA model trained on nucleobases, it is over DNA strings rather than amino-acid sequences. A model trained on cased text assigns probability to a lowercased string, but only as one casing among many. The mass that every other casing of the same text carries is left out of the score (Figure 1.1). In these settings, the strings a model produces and the strings an application needs differ, and the gap between them is bridged in practice by ad hoc post-processing. We refer to this gap throughout the thesis as the **string-mismatch problem**.

Figure 1.1 makes the gap concrete. Even for a target as short as the lowercase word `hello`, the set of GPT-2 BPE token sequences that map to it is large: any sequence whose lowercased concatenation matches `hello`

[1] Chapter 2 makes the notions used informally in this chapter (language models, alphabets, prefixes, transducers) precise.

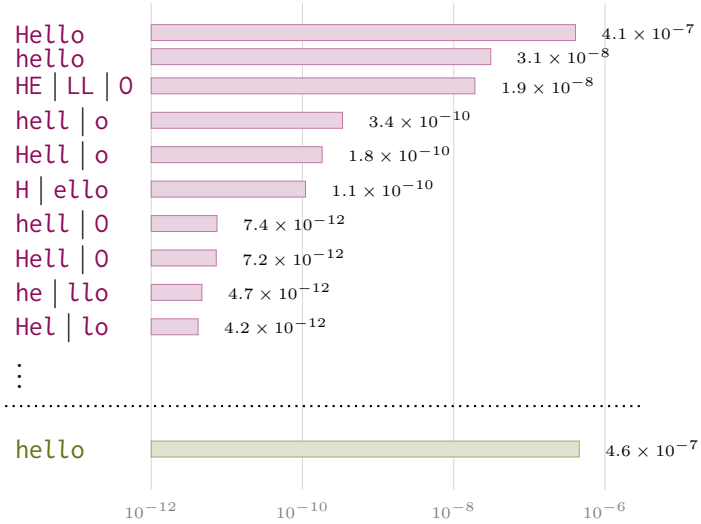


Figure 1.1: GPT-2 probabilities for the BPE token strings (boundaries marked with |) that, when lowercased, match hello. The aggregate probability of all such sequences appears at the bottom. Adapted from Snæbjarnarson et al. (2026a).

contributes mass. The target probability is therefore obtained by summing the source probabilities of all such token strings (total 4.6×10^{-7}), rather than reading off the probability of the single token string hello (3.1×10^{-8}).

The lowercase example is simple, but it already captures the difficulty: a summation over a preimage that grows with the target length. We now give two examples that recur throughout this thesis:

From byte-pair encodings to words. Modern large language models represent their inputs and outputs as sequences of subword tokens, often using vocabularies constructed with byte-pair encoding (BPE) (Gage, 1994; Sennrich, Haddow, and Birch, 2016). One and the same English sentence admits many encodings. Take

(1) *Dr. Lemaître was flabbergasted.* 🤯

Under GPT-4o's byte-pair encoding (OpenAI, 2024), ex. (1) becomes the sequence of subword tokens shown below. Token ids are shown beneath the corresponding tokens.

(2)
```
Dr  .  ␣L ema \C3\AEtre ␣was ␣fl ab berg asted .  ␣\F0\9F\A4 \AF
5822 13 451 4603     29135      673   1548 378  9667   23030  13      93643      107
```

The byte-pair segmentation is driven by substring frequency, not by linguistic

structure. This segmentation does not always match the units required downstream. For applications defined over words, probabilities must be evaluated at word boundaries rather than only at the boundaries imposed by the model's subword tokenizer. Under the Penn Treebank tokenization (PTB) (Marcus, Santorini, and Marcinkiewicz, 1993), for instance, ex. (1) becomes a sequence of word-level units:

(3) DR. LEMAÎTRE WAS FLABBERGASTED . 🤯

BPE units and PTB units do not line up in general. For the contraction *don't*, GPT-2's BPE yields the tokens `don` and `'t`, the Penn Treebank tokenizer yields the units `do` and `n't`, and a naïve whitespace splitter yields `don't`. More generally, when an analysis requires scoring of word-level units, token-level probabilities must be mapped onto those units. Prior work uses several aggregation conventions, differing especially in how probability mass at word boundaries is assigned (Nair and Resnik, 2023; Oh and Schuler, 2024; Pimentel and Meister, 2024; Wilcox et al., 2020). Other applications benefit from finer-grained representations (Ingólfsdóttir et al., 2023; Xue et al., 2022). Under UTF-8, the same utterance is represented by the following byte string:[2]

(4)
```
D r . ␣ L e m a \C3\AE t r e ␣ w a s
68 114 46 32 76 101 109 97 195 174 116 114 101 32 119 97 115
␣ f l a b b e r g a s t e d . ␣ \F0\9F\A4\AF
32 102 108 97 98 98 101 114 103 97 115 116 101 100 46 32 240 159 164 175
```

The same utterance may therefore be represented using BPE subwords, PTB words, characters, or bytes, depending on the application.

From DNA to amino acids. Genomic language models are trained on sequences over the four-letter nucleobase alphabet {`A`, `C`, `G`, `T`}. Many downstream uses (functional annotation, variant-effect prediction, mutation analysis) are phrased in terms of *amino acids*. The mapping from DNA to amino acids is deterministic and many-to-one because most amino acids are encoded by more than one three-nucleobase codon. The *mature oxytocin nonapeptide*, for example, can be encoded by a 27-nucleobase DNA sequence (ex. (5)) that translates codon-by-codon into the nine amino acids in ex. (6).

(5) `T G T T A C A T A C A A A A T T G T C C T C T A G G T`

(6) `C Y I Q N C P L G`

[2]The same rendered text may have more than one canonically equivalent Unicode representation and therefore more than one UTF-8 byte sequence. The character `î`, for instance, may be represented in composed form (`\C3\AE`) or decomposed form (`i\CC\82`). See `https://unicode.org/reports/tr15/`.

Scoring ex. (6) under a DNA-trained model means summing the probabilities of all DNA strings in its preimage, including ex. (5).

Even when the transformations look simple, the implied probability calculations are not. The conversion of subword tokens to bytes (ex. (4)) reduces to a lookup table on individual tokens, and the conversion of DNA to amino acids (ex. (6)) reduces to a fixed codon-to-amino-acid table. Yet in both cases the number of source-side strings that realize a given target prefix can grow exponentially with target length, and a faithful language-model conversion must sum over the preimage rather than picking a single representative. The PTB segmentation (ex. (3)) adds a different kind of difficulty: its rules depend on surrounding context, so whether a period belongs to the title Dr. or stands as its own sentence-final unit cannot be decided from the symbol alone.

In all of these examples, generating from a source language model and transforming each completed sample yields samples from the target distribution. Such methods, however, do not define a *language model interface* where the model can score a string, condition on a prefix, or produce a next-symbol distribution over the target alphabet.

1.2 Prior work

Language models assign probability to strings, but applications often benefit from probabilities over *equivalence classes* of strings. Examples include the token sequences that decode to one target sequence (as in Figure 1.1), the casings of one word, and the surface forms of equivalent answers. Several lines of work have encountered instances of this mismatch and developed targeted remedies, ranging from heuristic score adjustments to exact algorithms for restricted cases.

1.2.1 Problem-specific approaches

Word probabilities. A common need in NLP is deriving a word probability from a subword model. A straightforward approach computes a word's log-probability by summing the log-probabilities of its constituent subword tokens (Nair and Resnik, 2023; Wilcox et al., 2020). Pimentel and Meister (2024) and Oh and Schuler (2024) diagnose a defect in the usual computation for beginning-of-word tokenizers. Multiplying the probabilities of the tokens spelling a word does not condition on the word ending. For example, the token sequence spelling *mat* is also a prefix of sequences spelling *matron*; counting both as possible next words therefore counts overlapping rather than

disjoint events, and the resulting word probabilities may sum to more than one. Both papers address the missing end condition by assigning the following whitespace to the current word. Giulianelli et al. (2024) reject the claim that the trailing boundary must always belong to the preceding word. They argue that the region to be scored should instead follow the experimental design, and obtain its probability by marginalizing the token-level model into a character-level model.

These token-aggregation conventions are easy to implement, but they do not always define units consistently. Chapter 6 shows how a unit's identity can consequently depend on its position in the string and instead derives unit probabilities by transforming the model according to an experimentally chosen unit definition.

Marginalizing over tokenizations. The number of tokenizations of a text can grow exponentially with its length, but standard practice scores only the canonical one. Cao and Rimell (2021) argue that models should be evaluated on the marginal likelihood over tokenizations and find the gap largest out of domain, and Chirkova et al. (2023) find that the gap is small in most cases but grows on text with long complex words. Geh et al. (2024) show that the marginal is #P-hard to compute, that the most likely tokenization is NP-hard to find, and that noncanonical tokenizations carry usable signal. Phan et al. (2024) show that conditioning a language model on the canonical tokenization of a character prefix can give biased character-level predictions even when the token-level distribution is learned exactly and derive corrections for maximum-prefix and BPE tokenization. Vieira et al. (2025b) condition the model on producing only canonical BPE encodings rather than summing over alternative tokenizations.

The prompt boundary. The mismatch between a text prefix and its tokenization also arises at the boundary between a prompt and its continuation. For instance, appending a trailing space to a prompt can reroute its encoding onto a low-probability token path and thereby discard otherwise-likely continuations. This is known as the *prompt-boundary problem* (Vieira et al., 2025a). Existing approaches address this problem by backing up and re-constraining the first generated tokens through *token healing* (Lundberg and Ribeiro, 2023) or its generalization, *token alignment* (Athiwaratkun et al., 2024).

Answer scoring. In evaluation of NLP tasks, probability mass can split across surface forms of the same answer, so the highest-probability string need not be the most probable answer (Holtzman et al., 2021). Alternatives to raw

string probability include length-normalized scoring (Brown et al., 2020). To address surface-form competition specifically, Holtzman et al. (2021) propose domain-conditional PMI, which reweights each answer by its probability after a generic task-specific prompt. Both methods score each candidate separately. An even coarser shortcut ranks answer labels by their first-token probabilities; Wang et al. (2024) show that these rankings can disagree substantially with the answers instruction-tuned models produce as text. Semantic-entropy methods aggregate the probabilities of sampled strings within equivalence classes inferred from bidirectional entailment (Farquhar et al., 2024; Kuhn, Gal, and Farquhar, 2023).

Beyond natural language. The same problem arises beyond natural language when several strings represent the same object. In chemistry, a molecule corresponds to many SMILES strings.[3] Bjerrum (2017) average property predictions across enumerated SMILES for the same molecule. Arús-Pous et al. (2019) likewise aggregate across representations, but sum their sequence probabilities to obtain molecule-level likelihoods. That choice of representation can matter substantially: Runcie, Deane, and Imrie (2026) find that reasoning models perform worse on randomized SMILES representations than on canonical ones.

1.2.2 Model transformation and inference

The closest predecessors to this work convert token-level language models to character-level models by explicitly marginalizing over tokenizations. Two broader bodies of work supply the tools this thesis uses to generalize this idea: finite-state methods for representing transformations and Monte Carlo methods for estimating the resulting quantities.

Tokens to characters. Vieira et al. (2025a) turn a token-level language model into a character-level one. They compute character-prefix probabilities exactly by enumerating a finite cover and approximately with a pruned beam when the complete cover is too large. Their construction applies to any strict-prefix-monotone decoder and retains the probability of every token sequence, including noncanonical tokenizations. Related work obtains more efficient algorithms by treating only the tokenizer's canonical encoding of each string as valid. Phan et al. (2024) show that querying a language model after tokenizing a text prefix need not recover the corresponding next-character

[3]For example, ethanol can be represented as either `CCO` or `OCC`, depending on the traversal of its molecular graph.

distribution, and derive corrections for maximum-prefix and BPE tokenization. Phan et al. (2025) express byte probabilities as sums over canonically valid covers and apply the resulting interface to in-filling and model ensembles. For BPE, Hayase et al. (2026) show that the canonically valid paths covering a byte prefix share a fixed token prefix and differ only within a bounded suffix near the byte boundary. Their algorithm maintains this suffix incrementally and marginalizes over its branches.

This thesis generalizes these settings in three directions: transformations that need to look ahead before emitting (Chapter 3), transformations under which some source strings require string probabilities rather than prefix probabilities (the *remainder* introduced in Chapter 3), and unbiased estimation when the exact sum is out of reach (Chapter 5). Section 3.5 gives a more technical account of the relationship, and the algorithms of Chapter 4 reduce to the Vieira et al. algorithm in the special case.

Transducers in natural language processing. Transducers have a long history in language and speech processing (Mohri, 1997), and automata have long been used to structure decoding in statistical machine translation (Allauzen et al., 2014). More recently, finite-state machinery has been used to constrain neural sequence generation, for example by enforcing poetic form or restricting grammatical-error-correction hypotheses (Ghazvininejad et al., 2016; Stahlberg, Bryant, and Byrne, 2019). Related methods constrain the outputs of pretrained language models according to lexical or structural requirements (Cognetta et al., 2025; Koo, Liu, and He, 2024; Willard and Louf, 2023). Hamilton and Mimno (2025) further show that grammar-constrained classification can be sensitive to the chosen output representation, including whether permitted tokens carry leading whitespace. Subword tokenizers also admit finite-state implementations. WordPiece can be implemented with an Aho–Corasick-inspired trie (Song et al., 2021). BPE tokenization can be realized by a subsequential transducer (Berglund and Merwe, 2023), and the resulting valid token sequences can be recognized by a deterministic finite automaton (Berglund, Martens, and Merwe, 2024). Finite-state structure has also been combined with learned neural scoring, through context-dependent arc weights or path scores (Lin et al., 2019; Rastogi, Cotterell, and Eisner, 2016). Those approaches train the neural scorer as part of the transduction model, whereas the construction in this thesis applies a fixed transformation to the distribution defined by a separately trained language model.

Monte Carlo for language models. Sequential Monte Carlo (SMC) has recently been used to steer language-model generation toward distributions

defined by constraints (Lew et al., 2023; Lipkin et al., 2025; Loula et al., 2025; Xefteri et al., 2025; Zhao et al., 2024). Twisted variants use information about future targets to guide intermediate particle distributions (Heng et al., 2020; Whiteley and Lee, 2014). *Sampling without replacement*, which keeps particle or candidate sets distinct, has been used both in particle methods (Fearnhead and Clifford, 2003; Shah and Kroese, 2018) and in language-model decoding (Meister et al., 2021a). The estimator of Chapter 5 uses sampling without replacement within SMC to maintain distinct particles while estimating transduced-model prefix probabilities, with Horvitz–Thompson (Horvitz and Thompson, 1952) reweighting preserving each particle's weight in expectation.

1.3 Research questions

The central problem of this thesis is how to turn a pretrained language model into a new model—potentially by changing the strings it scores—without updating any parameters. Addressing it requires defining the transformed model and computing its autoregressive interface, exactly or approximately. This thesis asks three research questions.

RQ1 How can a deterministic string-to-string transformation turn a pretrained language model into a new language model? (Chapter 3)

RQ2 How can the transformed model's autoregressive interface be computed exactly when feasible and approximated otherwise? (Chapter 4)

RQ3 When exact computation is out of reach, can the transformed model's prefix probabilities be estimated without bias? (Chapter 5)

1.4 The TLM approach

We tackle the string-mismatch problem introduced in Section 1.1 by composing the model with the transformation itself. This is the familiar pattern of transforming a random variable: a string drawn from the source model, pushed through the map, is a draw from a new distribution, and this thesis treats that induced distribution as a language model in its own right. Given a language model $p_{\mathcal{X}}$ over a source alphabet $\mathcal{X}$ (formalized in Section 2.2) and a function $f\colon \mathcal{X}^* \to \mathcal{Y}^*$, the **pushforward** of $p_{\mathcal{X}}$ under f is a new language model $p_{\mathcal{Y}}$ over $\mathcal{Y}^*$, defined by summing over the preimage of each target string:

$$p_{\mathcal{Y}}(\boldsymbol{y}) \stackrel{\text{def}}{=} \Pr_{X \sim p_{\mathcal{X}}}[f(X) = \boldsymbol{y}] = \sum_{\boldsymbol{x} \in f^{-1}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}). \tag{1.1}$$

This sum can be intractable in general. Encoding f as a finite-state transducer—a state machine that reads source strings and emits target strings—lets the

algorithms in Chapters 4 and 5 compute or estimate this sum. Transducers capture many of the transformations we care about, including conversions between tokenizations, normalization, and codon tables. We call the resulting object a **transduced language model** (TLM). A TLM transforms the distribution defined by an already-trained model $p_{\mathcal{X}}$ without updating its parameters. The transduced model retains the usual language-model interface: it can be scored, conditioned on, and sampled from over the target alphabet. With this construction, the alphabet and distribution exposed to an application need not be those fixed during training.

1.5 Contributions and roadmap

The substantive contributions, grouped by chapter, are:

Chapter 3: Transduced language models (RQ1). We formalize the construction of a language model from a source language model and a string-to-string transformation encoded by a finite-state transducer. We introduce the *precover* of a target prefix and its *quotient–remainder decomposition*, and give sufficient conditions under which this decomposition is finite.

Chapter 4: Inference (RQ2). We give an exact algorithm for prefix probabilities under a TLM based on the decomposition of Chapter 3, together with optimizations based on input-projection universality and next-symbol decomposition shortcuts. When exhaustive enumeration is impractical, a pruning fallback makes the computation feasible at the cost of systematic downward bias.

Chapter 5: Unbiased inference via stochastic estimation (RQ3). To remove the downward bias introduced by the pruning fallback in Chapter 4, we develop a sequential Monte Carlo estimator built on sampling without replacement and corrected by Horvitz–Thompson reweighting. The estimator is unbiased and its variance decreases empirically as the number of particles used grows. At a fixed maximum particle count in the DNA-to-amino-acid experiment, running time grows approximately linearly over tested target lengths even as the compatible source-prefix set grows exponentially.

Chapter 6: Surprisal in psycholinguistics. Surprisal theory (Hale, 2001; Levy, 2008) predicts human reading times from the negative log-probability of upcoming linguistic units. We separate two modeling choices often conflated in practice: the *unit inventory* over which surprisal is defined and the *region*

of interest over which it is aggregated. We formalize these choices and convert pretrained byte-pair-encoded LMs into TLMs over different unit inventories. Experiments on the MECO eye-tracking corpus evaluate four unit choices, each with its own observations and baseline predictors.

Chapter 2

Background

This chapter collects preliminary material reproduced or adapted from Snæbjarnarson et al. (2026a) and Kiegeland et al. (2026).

This chapter collects the shared formal preliminaries that the body chapters rely on. The notation conventions are those of the Notation pages in the frontmatter (page xxii). The Monte Carlo machinery of Chapter 5 (importance sampling, resampling, proper weighting) is deliberately held until that chapter, where it is introduced along with the algorithms that use it.

2.1 Strings, alphabets, and languages

Let $\mathcal{X}$ be an **alphabet**, i.e., a finite, non-empty set whose elements are called **symbols**. Let $\mathcal{X}^*$ denote the set of all finite strings over $\mathcal{X}$, including the empty string $\varepsilon_{\mathcal{X}}$. When there is no risk of ambiguity with other alphabets, we simply write ε. We use $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}^*$ to denote strings, and $Z, Z' \subseteq \mathcal{X}^*$ to denote sets of strings. The **concatenation** of two strings is written $\boldsymbol{x}\boldsymbol{x}'$, and concatenation extends to sets by $ZZ' \stackrel{\text{def}}{=} \{\boldsymbol{x}\boldsymbol{x}' \mid \boldsymbol{x} \in Z, \boldsymbol{x}' \in Z'\}$. In the singleton case, $\boldsymbol{x}Z' \stackrel{\text{def}}{=} \{\boldsymbol{x}\}Z'$ and $Z\boldsymbol{x}' \stackrel{\text{def}}{=} Z\{\boldsymbol{x}'\}$. We write $\boldsymbol{x} \preceq \boldsymbol{x}'$ when $\boldsymbol{x}$ is a **prefix** of $\boldsymbol{x}'$, and $\boldsymbol{x} \prec \boldsymbol{x}'$ when it is a **strict prefix**. Conversely, if $\boldsymbol{x} \preceq \boldsymbol{x}'$, we say $\boldsymbol{x}'$ is an **extension** of $\boldsymbol{x}$. A **language** is any subset of $\mathcal{X}^*$.

2.2 Language models

A **language model** $p_{\mathcal{X}}$ is a probability distribution over a set of strings $\mathcal{X}^*$. Let $\textsc{eos} \notin \mathcal{X}$ be a special **end-of-string symbol**. We define the **prefix probability** of $p_{\mathcal{X}}$ as the probability that a string $X \sim p_{\mathcal{X}}$ starts with a

given prefix $\boldsymbol{x}$,

$$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}' \in \mathcal{X}^*} p_{\mathcal{X}}(\boldsymbol{x}\boldsymbol{x}'), \tag{2.1}$$

and the **conditional prefix probability**,

$$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x}) \stackrel{\text{def}}{=} \frac{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}\boldsymbol{x}')}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})}, \tag{2.2}$$

$$\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \stackrel{\text{def}}{=} \frac{p_{\mathcal{X}}(\boldsymbol{x})}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})}, \tag{2.3}$$

when $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) > 0$, and otherwise we set $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x}) \stackrel{\text{def}}{=} 0$ and $\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \stackrel{\text{def}}{=} 1$. Therefore, $\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$ is a probability distribution over $\mathcal{X} \sqcup \{\text{EOS}\}$ for all $\boldsymbol{x} \in \mathcal{X}^*$. Using this structure, any language model $p_{\mathcal{X}}$ may be factorized as

$$p_{\mathcal{X}}(\boldsymbol{x}) = \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \prod_{t=1}^{|\boldsymbol{x}|} \overrightarrow{p_{\mathcal{X}}}(x_t \mid \boldsymbol{x}_{<t}). \tag{2.4}$$

This factorization defines a left-to-right generative process: starting from $\boldsymbol{x} = \varepsilon$, we repeatedly sample $x' \sim \overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$. If $x' = \text{EOS}$, we stop, and otherwise we update $\boldsymbol{x}$ to $\boldsymbol{x}x'$. Conditional generation simply starts from the conditioning prefix instead of the empty string. We refer to the quantities in Equations (2.1) to (2.3) as the **autoregressive interface** to the language model $p_{\mathcal{X}}$.

In practice, an autoregressive model specifies the conditionals directly, and the product in Equation (2.4) recovers a distribution over $\mathcal{X}^*$ exactly when the conditionals are **tight**: their generative process reaches EOS with probability one (Cotterell et al., 2024). We assume tightness throughout, which equates a model's conditionals with the prefix conditionals above. A language model then places all its mass on finite strings, so the mass beyond depth n, $\sum_{|\boldsymbol{x}|=n} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$, vanishes as n grows, the property the halting arguments of Chapter 5 rely on.

Example 2.1 — Prefix probabilities

Let $\mathcal{X} = \{\mathsf{a}, \mathsf{b}\}$ and let $p_{\mathcal{X}}$ put all its mass on three strings:

$$p_{\mathcal{X}}(\mathsf{a}) = 0.2, \qquad p_{\mathcal{X}}(\mathsf{ab}) = 0.3, \qquad p_{\mathcal{X}}(\mathsf{b}) = 0.5.$$

Prefix probabilities sum over extensions (Equation (2.1)):

$$\overrightarrow{p_{\mathcal{X}}}(\varepsilon) = 1, \qquad \overrightarrow{p_{\mathcal{X}}}(\mathsf{a}) = p_{\mathcal{X}}(\mathsf{a}) + p_{\mathcal{X}}(\mathsf{ab}) = 0.5,$$
$$\overrightarrow{p_{\mathcal{X}}}(\mathsf{b}) = 0.5, \qquad \overrightarrow{p_{\mathcal{X}}}(\mathsf{ab}) = 0.3.$$

These prefix probabilities determine the full autoregressive interface. At the empty context,

$$\overrightarrow{p_{\mathcal{X}}}(\mathsf{a} \mid \varepsilon) = \overrightarrow{p_{\mathcal{X}}}(\mathsf{b} \mid \varepsilon) = 0.5, \qquad \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \varepsilon) = \frac{p_{\mathcal{X}}(\varepsilon)}{\overrightarrow{p_{\mathcal{X}}}(\varepsilon)} = 0,$$

and after reading a,

$$\overrightarrow{p_{\mathcal{X}}}(\mathsf{b} \mid \mathsf{a}) = \frac{\overrightarrow{p_{\mathcal{X}}}(\mathsf{ab})}{\overrightarrow{p_{\mathcal{X}}}(\mathsf{a})} = \frac{0.3}{0.5} = 0.6, \qquad \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \mathsf{a}) = \frac{p_{\mathcal{X}}(\mathsf{a})}{\overrightarrow{p_{\mathcal{X}}}(\mathsf{a})} = \frac{0.2}{0.5} = 0.4.$$

The end-of-string symbol receives exactly the probability of stopping after the current prefix, and each conditional distribution sums to one.

The factorization (Equation (2.4)) checks out:

$$p_{\mathcal{X}}(\mathsf{ab}) = \underbrace{\overrightarrow{p_{\mathcal{X}}}(\mathsf{a} \mid \varepsilon)}_{0.5} \cdot \underbrace{\overrightarrow{p_{\mathcal{X}}}(\mathsf{b} \mid \mathsf{a})}_{0.6} \cdot \underbrace{\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \mathsf{ab})}_{1} = 0.3.$$

2.3 Cylindrical sets

A cylindrical set contains every extension of each of its elements. Let $Z, Z' \subseteq \mathcal{X}^*$. We define the **cylinder** over Z as $\langle Z \rangle \stackrel{\text{def}}{=} Z\mathcal{X}^*$, and we say that Z is **cylindrical** if $Z = \langle Z \rangle$. The union of cylinder sets is again a cylinder set, since cylinders are upward-closed. We define the **basic cylinder** for $\boldsymbol{x}$ as $\langle \boldsymbol{x} \rangle \stackrel{\text{def}}{=} \langle \{\boldsymbol{x}\} \rangle$. The **prefix-base** operation $\text{pf}(Z)$ is defined by $\text{pf}(Z) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in Z : \nexists\, \boldsymbol{x}' \in Z \colon \boldsymbol{x}' \prec \boldsymbol{x}\}$. This operation uniquely partitions $\langle Z \rangle$ into basic cylinders over $\text{pf}(Z)$. We say that Z is **prefix-free** if $\text{pf}(Z) = Z$. Since $\text{pf}(Z)$ is prefix-free, the basic cylinders $\{\langle \boldsymbol{x} \rangle \mid \boldsymbol{x} \in \text{pf}(Z)\}$ are pairwise disjoint, and $\langle Z \rangle = \bigsqcup_{\boldsymbol{x} \in \text{pf}(Z)} \langle \boldsymbol{x} \rangle$.

Example 2.2 — Cylinders and the prefix-base

Let $\mathcal{X} = \{\mathsf{a}, \mathsf{b}\}$ and consider

$$Z = \{\mathsf{a},\ \mathsf{ab},\ \mathsf{ba}\}.$$

The cylinder $\langle Z \rangle$ collects every extension of every element, but redundantly: every extension of ab is already an extension of a,

$$\langle \mathsf{ab} \rangle \subseteq \langle \mathsf{a} \rangle.$$

The prefix-base removes exactly this redundancy, dropping ab because its proper prefix a is in the set:

$$\mathrm{pf}(Z) = \{\mathsf{a},\ \mathsf{ba}\}.$$

What remains is prefix-free, so the basic cylinders are disjoint (ba starts with b) and partition the original cylinder:

$$\langle Z \rangle = \langle \mathsf{a} \rangle \sqcup \langle \mathsf{ba} \rangle.$$

Because cylinders over a prefix-free set are disjoint, their probabilities can be added without double counting. This is why the quotient of Chapter 3 is defined to be prefix-free, and why prefix probabilities of quotient elements and exact probabilities of remainder elements together sum to the prefix probability of the transduced model.

2.4 Finite-state transducers[1]

A **transducer** is a state machine that encodes a string-to-string relation $f \subseteq \mathcal{X}^* \times \mathcal{Y}^*$. Expressing a relation as a transducer exposes the computational structure needed to develop efficient algorithms. Formally, a **finite-state transducer**[2] (**FST**) f is a tuple $(\mathsf{S}, \mathcal{X}, \mathcal{Y}, \mathsf{I}, \mathsf{F}, \mathsf{T})$, where S is a finite set of **states** and $\mathcal{X}$, $\mathcal{Y}$ are alphabets of **input** and **output** symbols. The sets $\mathsf{I}, \mathsf{F} \subseteq \mathsf{S}$ are the **initial** and **accepting** states. $\mathsf{T} \subseteq \mathsf{S} \times (\mathcal{X} \cup \{\varepsilon\}) \times (\mathcal{Y} \cup \{\varepsilon\}) \times \mathsf{S}$ is a set of **transitions**. We render a transition $(s, x, y, s') \in \mathsf{T}$ as $s \xrightarrow{x:y} s'$. We say the transition **scans** x and **emits** y. We write $\mathsf{T}(s)$ for the set of outgoing transitions from state s, and $\mathsf{T}(s, x)$ for those that scan x.

The transducer defines a set of **paths** Π. Each **path** $\boldsymbol{\pi} \in \Pi$ is a sequence $s_0 \xrightarrow{x_1:y_1} s_1 \cdots s_{N-1} \xrightarrow{x_N:y_N} s_N$. We call $\boldsymbol{\pi}$ an **accepting path** if $s_0 \in \mathsf{I}$ and $s_N \in \mathsf{F}$. For a path as above, let $\boldsymbol{x} = x_1 \cdots x_N$ and $\boldsymbol{y} = y_1 \cdots y_N$ denote the strings it scans and emits, respectively. The **relation defined** by f is

$$[\![\mathsf{f}]\!] \stackrel{\text{def}}{=} \{(\boldsymbol{x}, \boldsymbol{y}) \mid s_0 \xrightarrow{x_1:y_1} s_1 \cdots s_{N-1} \xrightarrow{x_N:y_N} s_N \in \Pi \colon s_0 \in \mathsf{I}, s_N \in \mathsf{F}\}, \tag{2.5}$$

[1] In Snæbjarnarson et al. (2026a), parts of this material appear in an appendix.
[2] See Pin (2021, Ch. 2 & 3) for a detailed treatment.

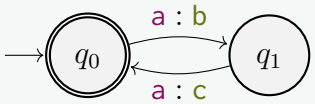
i.e., each accepting path contributes (not necessarily uniquely) a pair of scanned and emitted strings.

When every transition of a transducer scans and emits the same symbol ($x = y$), the machine acts as a **finite-state automaton** (**acceptor**) that recognizes a language $L \subseteq \mathcal{X}^*$, the set of strings admitted by at least one accepting path. Such a machine is a **nondeterministic finite automaton** (**NFA**) in general. It is a **deterministic finite automaton** (**DFA**) if it has a single initial state, no ε-transitions, and at most one transition per state and input symbol. Every NFA can be converted to an equivalent DFA by **determinization** (Rabin and Scott, 1959).

In diagrams, an arrow with no source state marks an initial state, a double-lined circle marks an accepting state, and a transition label $x{:}y$ gives the scanned and emitted symbols, with ε for neither-scanning nor-emitting sides.

Example 2.3 — A parity transducer

Consider the input-deterministic (one arc per state and input symbol) two-state transducer over $\mathcal{X} = \{\mathsf{a}\}$ and $\mathcal{Y} = \{\mathsf{b}, \mathsf{c}\}$:



Reading aaaa traces the unique path $q_0 \xrightarrow{\mathsf{a}:\mathsf{b}} q_1 \xrightarrow{\mathsf{a}:\mathsf{c}} q_0 \xrightarrow{\mathsf{a}:\mathsf{b}} q_1 \xrightarrow{\mathsf{a}:\mathsf{c}} q_0$, which scans aaaa, emits bcbc, and ends in the accepting state q_0. Reading aaa strands the machine in the non-accepting q_1. The relation defined is therefore

$$[\![\mathsf{f}]\!] = \{(\mathsf{a}^{2n}, (\mathsf{bc})^n) \colon n \geq 0\},$$

a function whose domain is the even-length strings: the machine computes its output and enforces a parity constraint at once.

2.4.1 Transducer variants

We say a transducer is **functional** if it defines a function, and **partially functional** if it defines a partial function. A transducer is **input-deterministic** if $|\mathsf{I}| = 1$ and, for every state $s \in \mathsf{S}$, $|\mathsf{T}(s, \varepsilon)| = 0$ and $|\mathsf{T}(s, x)| \leq 1$ for all $x \in \mathcal{X}$.[3] For input-deterministic transducers, each source string $\boldsymbol{x} \in \mathcal{X}^*$ has at most one accepting path that scans $\boldsymbol{x}$ and, therefore, can emit at most one

[3]The more general notion of a *subsequential transducer* also includes a final output function. This distinction does not arise for copy-transducers.

target string $\boldsymbol{y} \in \mathcal{Y}^*$. Thus, every input-deterministic transducer defines a (partial) function. A **copy-transducer** is one where every transition emits the same symbol as it scans, i.e., every transition is of the form $s \xrightarrow{x:x} s'$. We abbreviate copy transitions as $s \xrightarrow{x} s'$. Copy-transducers define partial identity functions: they map each string in a designated subset to itself, and drop all others. When context is clear, we identify the relation defined by a copy-transducer with the language it accepts.

2.4.2 Operations

Transducers support **composition**: given transducers $\mathtt{f}$ and $\mathtt{f}'$, their composition $\mathtt{f} \circ \mathtt{f}'$ is a transducer denoting $[\![\mathtt{f} \circ \mathtt{f}']\!] \stackrel{\text{def}}{=} [\![\mathtt{f}]\!] \circ [\![\mathtt{f}']\!]$.[4] Mohri (2009, sec. 5.1) gives an efficient composition algorithm. We denote a transducer encoding the relation $[\![\mathtt{f}]\!] \circ \{(\boldsymbol{y}', \boldsymbol{y}') \mid \boldsymbol{y}' \in \langle \boldsymbol{y} \rangle\}$ as $\mathtt{f} \circ \boldsymbol{y}\mathcal{Y}^*$, and freely coerce between these representations when intent is clear from context.

The **input projection** operation $\mathsf{proj}_{\mathcal{X}}(\mathtt{f}) \stackrel{\text{def}}{=} (\mathsf{S}, \mathcal{X}, \mathcal{X}, \mathsf{I}, \mathsf{F}, \{(s \xrightarrow{x:x} s') \mid s \xrightarrow{x:y} s' \in \mathsf{T}\})$ yields a copy-transducer encoding $[\![\mathsf{proj}_{\mathcal{X}}(\mathtt{f})]\!] = \{(\boldsymbol{x}, \boldsymbol{x}) \mid \exists\, \boldsymbol{y} \colon (\boldsymbol{x}, \boldsymbol{y}) \in [\![\mathtt{f}]\!]\}$.

Let $\mathtt{f}_{[s]}$ denote the operation of **force-starting** $\mathtt{f}$ in state s, $\mathtt{f}_{[s]} \stackrel{\text{def}}{=} (\mathsf{S}, \mathcal{X}, \mathcal{Y}, \{s\}, \mathsf{F}, \mathsf{T})$. This yields a machine defining the set of source–target suffix pairs generated by paths starting at s and ending in an accepting state.

A state s is **input-projection universal** (**IP-universal**) if $[\![\mathsf{proj}_{\mathcal{X}}(\mathtt{f}_{[s]})]\!] = \mathcal{X}^*$: whatever input follows, the machine can still scan it to acceptance. We write $\mathsf{U} \subseteq \mathsf{S}$ for a set of IP-universal states found by the sufficient test in Section 4.3.3.

We use $\mathsf{trim}(\mathtt{f})$ to denote a **trimming** operation that removes all states and edges not on any accepting path, and $\mathsf{determinize}(\mathtt{f})$ to denote a transformation that maps a (partially) functional transducer to an equivalent input-deterministic one. Such a mapping is not always realizable with a finite number of states.[5] For copy-transducers, however, input-determinization is always possible, but may result in exponential blowup in the worst case.[6] These are standard FST operations (Pin, 2021, 2025), available in libraries such as Pynini (Gorman, 2016). Example 4.2 in Chapter 4 exercises composition, projection, determinization, and trimming step by step, building the

[4] Here, **relation composition** is given by $f \circ g \stackrel{\text{def}}{=} \{(x, z) \mid \exists\, y \colon (x, y) \in f, (y, z) \in g\}$.

[5] Choffrut (1977) gives such an algorithm, by using a power-set construction on the input side analogous to NFA determinization. He shows it yields a finite machine if and only if the represented function has bounded variation, and gives a testable condition known as the twinning condition.

[6] For readers familiar with finite-state automata, input-determinization of a copy-transducer is isomorphic to the determinization of an equivalent finite-state automaton.

machine on which that chapter's algorithms run.

2.4.3 Limitations of transducers

Finite-state transducers define the class of *rational relations* (Berstel, 1979, Ch. III). Because FSTs have only finitely many states, they cannot perform transformations that require unbounded memory. The transformations studied in this thesis all stay within the rational class.

Chapter 3

Transduced language models

This chapter contains material reproduced and adapted from Snæbjarnarson et al. (2026a), with editorial changes for narrative consistency.

Section 3.1 defines a transduced language model as the pushforward of a source model through a transducer and writes its prefix probability as a sum of source mass over a *precover*. That sum ranges over a set that can be infinite, so Section 3.2 decomposes the precover into a prefix-free *quotient* of cylinders and a *remainder*. Section 3.3 gives sufficient conditions under which both sets are finite, thereby turning the sum into a finite computation. Section 3.4 illustrates finite and infinite decompositions with three small machines, and Section 3.5 situates the construction against prior work. Chapter 4 develops algorithms for computing and using the decomposition.

3.1 The pushforward construction

Let $\mathcal{X}$ and $\mathcal{Y}$ be finite alphabets, let $p_{\mathcal{X}}$ be a language model over $\mathcal{X}^*$ (Section 2.2), and let $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ be a string-to-string function realized by a finite-state transducer $\mathtt{f}$ (Section 2.4). A **transduced language model** $p_{\mathcal{Y}}$ arises from applying f to a string drawn from $p_{\mathcal{X}}$: if $X \sim p_{\mathcal{X}}$, then $f(X)$ has the following probability mass function:

$$p_{\mathcal{Y}}(\boldsymbol{y}) \stackrel{\text{def}}{=} \Pr_{X \sim p_{\mathcal{X}}}[\boldsymbol{y} = f(X)] = \sum_{\boldsymbol{x} \in f^{-1}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \tag{3.1}$$

where $f^{-1}(\boldsymbol{y})$ is the **preimage** of $\boldsymbol{y}$, $f^{-1}(\boldsymbol{y}) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathcal{X}^* \colon \boldsymbol{y} = f(\boldsymbol{x})\}$. Put differently, in Equation (3.1), we sum over the strings $\boldsymbol{x}$ such that $f(\boldsymbol{x}) = \boldsymbol{y}$. Unfortunately, evaluating $p_{\mathcal{Y}}(\boldsymbol{y})$ exactly using Equation (3.1) is generally infeasible (since the preimage $f^{-1}(\boldsymbol{y})$ can be infinite), even though exact

sampling from $p_{\mathcal{Y}}$ is efficient (by sampling $\boldsymbol{x} \sim p_{\mathcal{X}}$ and applying f). Like all language models, a transduced language model $p_{\mathcal{Y}}$ has prefix and conditional prefix probability functions. Its prefix probability is

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}) = \Pr_{X \sim p_{\mathcal{X}}}[\boldsymbol{y} \preceq f(X)] = \sum_{\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \tag{3.2}$$

where $\mathcal{P}(\boldsymbol{y})$ is the **precover** of $\boldsymbol{y}$, with respect to f, defined as[1]

$$\mathcal{P}(\boldsymbol{y}) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathcal{X}^* : \boldsymbol{y} \preceq f(\boldsymbol{x})\}. \tag{3.3}$$

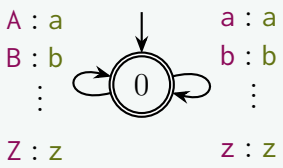
Prefix probabilities yield a conditional factorization of string probability (see Section 2.2), enabling efficient left-to-right autoregressive generation. In Section 3.2, we develop a method that computes the sum in Equation (3.2) in *finite* time for a general class of mappings, such as the lowercasing and DNA-to-amino-acid examples mentioned in the introduction.

3.2 Decomposing the precover

In Section 3.1, we saw that if we can sum over the precover of $\boldsymbol{y}$, we can calculate $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$ (Equation (3.2)), unlocking an autoregressive interface to the transduced language model. The following two examples illustrate how we can often compute this infinite sum by exploiting structural properties of the transducer.

Example 3.1 — The lowercasing transducer

The transducer below lowercases a string. For the target string ab, the precover is the infinite set $\mathcal{P}(\mathsf{ab}) = \langle\{\mathsf{AB}, \mathsf{Ab}, \mathsf{aB}, \mathsf{ab}\}\rangle$. Given a model $p_{\mathcal{X}}$ over the input language, the derivation below (3.4a–3.4c) applies Equation (3.2) to express $\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ab})$ as a sum of four source-language prefix probabilities:



[1] Note that the precover depends on f. We suppress this dependency when it is clear from context.

$$\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ab}) = \sum_{\boldsymbol{x} \in \mathcal{P}(\mathsf{ab})} p_{\mathcal{X}}(\boldsymbol{x}) \tag{3.4a}$$

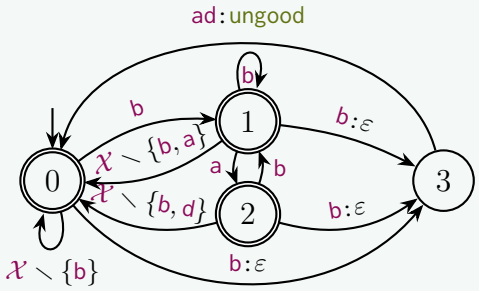
$$= \sum_{\boldsymbol{x}' \in \{\mathsf{AB},\mathsf{Ab},\mathsf{aB},\mathsf{ab}\}} \sum_{\boldsymbol{x} \in \langle \boldsymbol{x}' \rangle} p_{\mathcal{X}}(\boldsymbol{x}) \tag{3.4b}$$

$$= \overrightarrow{p_{\mathcal{X}}}(\mathsf{AB}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{Ab}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{aB}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{ab}) \tag{3.4c}$$

In Example 3.1, each input symbol maps to exactly one output symbol, so the precover decomposes neatly into cylinders. The next example shows what happens when some source strings cover the target string, while some of their extensions do not.

Example 3.2 — A remainder in Newspeak

The transducer below implements a *Newspeak*[a] rewrite rule: the word bad is replaced by ungood. For the target string ba, the cylinder $\langle \mathsf{bad} \rangle$ does not contribute to $\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ba})$ because bad does not map to a string prefixed by ba. The derivation below (3.5a–3.5c) shows how this cylinder is excluded.[b]



$$\overrightarrow{p_{\mathcal{Y}}}(\mathsf{ba}) = \sum_{\boldsymbol{x} \in \langle \mathsf{ba} \rangle \setminus \langle \mathsf{bad} \rangle} p_{\mathcal{X}}(\boldsymbol{x}) \tag{3.5a}$$

$$= \sum_{\boldsymbol{x} \in \bigcup_{x \in \mathcal{X} \setminus \{\mathsf{d}\}} \langle \mathsf{ba}x \rangle \cup \{\mathsf{ba}\}} p_{\mathcal{X}}(\boldsymbol{x}) \tag{3.5b}$$

$$= p_{\mathcal{X}}(\mathsf{ba}) + \sum_{x \in \mathcal{X} \setminus \{\mathsf{d}\}} \overrightarrow{p_{\mathcal{X}}}(\mathsf{ba}x) \tag{3.5c}$$

Because $\mathsf{bad} \in \langle \mathsf{ba} \rangle$ but $\mathsf{bad} \notin \mathcal{P}(\mathsf{ba})$, the precover cannot be decomposed

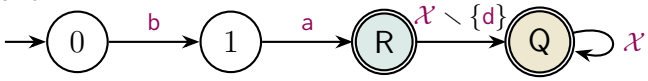
[a] *Newspeak* is the controlled language from Orwell's *1984* (Orwell, 1949). A complete Newspeak transducer would contain many more rules.

[b] Arcs labeled by a single symbol are *copy transitions*, e.g., b is shorthand for b : b, and a set label such as $\mathcal{X} \setminus \{\mathsf{b}\}$ denotes one copy-transition arc per symbol in the set.

entirely into cylinders. Instead, we decompose it into two disjoint parts: a maximal cylindrical subset and its complement in the precover. The *quotient* contains the shortest prefixes whose cylinders form the maximal cylindrical subset. The complement is the *remainder*. The final step (3.5c) illustrates a **computational shortcut**. For any $\boldsymbol{y}$ we can decompose $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$:

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}) = \underbrace{\sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})}_{\text{Quotient}} + \underbrace{\sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x})}_{\text{Remainder}} \tag{3.6}$$

The precover can also be represented as a finite-state automaton (the *precover machine*, constructed in general in Section 4.2.1) as shown below for $\mathcal{P}(\mathsf{ba})$. The single string accepted at R (teal node) marks the remainder element ba, and the state Q (yellow node) accepts $\langle \mathsf{ba} \cdot (\mathcal{X} \setminus \{\mathsf{d}\}) \rangle$ and marks the cylinder over the quotient.



The prefix decomposition of the precover. Let $f \colon \mathcal{X}^* \to \mathcal{Y}^*$ be a map. For each $\boldsymbol{y} \in \mathcal{Y}^*$, define $\mathcal{C}(\boldsymbol{y}) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathcal{X}^* \colon \langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})\}$. This set is a cylinder: if $\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y})$ then every extension of $\boldsymbol{x}$ also belongs to $\mathcal{C}(\boldsymbol{y})$. It is the largest cylinder contained in $\mathcal{P}(\boldsymbol{y})$. We define the **quotient** and **remainder** of $\boldsymbol{y}$ with respect to f (the prefix-base operation pf is illustrated in Example 2.2) as

$$\mathcal{Q}(\boldsymbol{y}) \stackrel{\text{def}}{=} \text{pf}(\mathcal{C}(\boldsymbol{y})) \qquad \text{and} \qquad \mathcal{R}(\boldsymbol{y}) \stackrel{\text{def}}{=} \mathcal{P}(\boldsymbol{y}) \setminus \mathcal{C}(\boldsymbol{y}) \tag{3.7}$$

We call the pair $(\mathcal{Q}(\boldsymbol{y}), \mathcal{R}(\boldsymbol{y}))$ the **optimal prefix decomposition** of $\mathcal{P}(\boldsymbol{y})$, characterized by three conditions: (i) $\mathcal{Q}(\boldsymbol{y})$ is *prefix-free*, (ii) $\mathcal{P}(\boldsymbol{y}) = \langle \mathcal{Q}(\boldsymbol{y}) \rangle \sqcup \mathcal{R}(\boldsymbol{y})$ (*validity*), and (iii) $\langle \mathcal{Q}(\boldsymbol{y}) \rangle = \mathcal{C}(\boldsymbol{y})$ (*maximality*): $\mathcal{Q}(\boldsymbol{y})$ identifies the largest cylinder in $\mathcal{P}(\boldsymbol{y})$. Instantiated on Example 3.2: $\mathcal{Q}(\mathsf{ba}) = \{\mathsf{ba}x \colon x \in \mathcal{X} \setminus \{\mathsf{d}\}\}$ and $\mathcal{R}(\mathsf{ba}) = \{\mathsf{ba}\}$, which is what Equation (3.5c) summed. This decomposition lets us compute prefix probabilities using the shortcut:

$$\begin{aligned}
\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}) &= \sum_{\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \\
&= \sum_{\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y}) \sqcup \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \\
&= \sum_{\substack{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}) \\ \boldsymbol{x}' \in \mathcal{X}^*}} p_{\mathcal{X}}(\boldsymbol{x}\boldsymbol{x}') + \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \\
&= \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) + \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x})
\end{aligned} \tag{3.8}$$

3.3 Conditions for finite decompositions[2]

The prefix-probability shortcut of Equation (3.6) evaluates a sum over the quotient $\mathcal{Q}(\boldsymbol{y})$ and a sum over the remainder $\mathcal{R}(\boldsymbol{y})$, so it reduces to a finite computation exactly when both sets are finite. We first define the relevant monotonicity properties of f, then restate the guarantee of previous work for strict-prefix monotone maps (Vieira et al., 2025a, Props 1 and 3; proposition 3.2), and finally give transducer-level conditions under which the quotient and remainder are finite even without monotonicity.

We say that a map $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ is **strict-prefix monotone** if and only if $\boldsymbol{x} \prec \boldsymbol{x}' \implies f(\boldsymbol{x}) \prec f(\boldsymbol{x}')$. Similarly f is **prefix monotone** if and only if $\boldsymbol{x} \preceq \boldsymbol{x}' \implies f(\boldsymbol{x}) \preceq f(\boldsymbol{x}')$. We say that a map f is **prefix-continuous** if, for every $\boldsymbol{y} \in \mathcal{Y}^*$, the set $\mathcal{P}(\boldsymbol{y})$ is cylindrical, i.e., $\mathcal{R}(\boldsymbol{y}) = \emptyset$. For a total f, these properties are related as follows.

$$\text{strict-prefix monotone} \implies \text{prefix monotone} \iff \text{prefix-continuous},$$

where the first implication is immediate (strict-prefix monotonicity covers the case $\boldsymbol{x} \prec \boldsymbol{x}'$, and $\boldsymbol{x} = \boldsymbol{x}'$ is trivial) and the equivalence is proposition 3.1 below. Totality of f matters because a partial map can be prefix monotone on its domain yet leave a nonempty remainder. The cylinder condition quantifies over every extension in $\mathcal{X}^*$, and an extension can leave the domain, as the machine of Example 3.4 shows. The first implication is strict, as the DNA-to-amino-acid map of Example 3.3 shows.

Example 3.3 — Monotonicity and the prefix decomposition

The examples show how the monotonicity properties affect the decomposition.

- *Lowercasing* (Example 3.1) is strict-prefix monotone: every symbol read emits exactly one symbol, so a strictly longer input always yields a strictly longer output.
- The *DNA-to-amino-acid map*, whose transducer is partially shown in Figure A.1, is prefix monotone but *not* strict. Writing f for the map that translates complete codons and ignores a trailing incomplete codon, we have

  $$f(\mathsf{A}) = f(\mathsf{AA}) = \varepsilon \quad \text{while} \quad \mathsf{A} \prec \mathsf{AA}.$$

  The output does not grow, violating strictness. But output, once produced, is never retracted, so $\boldsymbol{x} \preceq \boldsymbol{x}'$ still implies $f(\boldsymbol{x}) \preceq f(\boldsymbol{x}')$.

[2] In Snæbjarnarson et al. (2026a), the proofs of this section appear in an appendix.

This is why the map requires multi-symbol *lookahead*. After reading AA, it has not yet committed to an output symbol. The third base determines the output symbol.

- *Newspeak* (Example 3.2) is not even prefix monotone:

$$\mathsf{ba} \prec \mathsf{bad} \quad \text{but} \quad f(\mathsf{bad}) = \mathsf{ungood} \text{ does not extend } f(\mathsf{ba}) = \mathsf{ba}.$$

The monotonicity properties have the following consequences for the decomposition.

transformation	strict	pref. monotone	consequence
Lowercasing (Example 3.1)	✓	✓	$\mathcal{Q}(\boldsymbol{y})$ finite and $\mathcal{R}(\boldsymbol{y}) = \emptyset$ (proposition 3.2)
DNA to amino acids (Figure A.1)	✗	✓	$\mathcal{R}(\boldsymbol{y}) = \emptyset$ (proposition 3.1)
Newspeak rewrite (Example 3.2)	✗	✗	$\mathsf{ba} \in \mathcal{R}(\mathsf{ba})$

In the final row, ba belongs to the remainder because it covers the target, while its extension bad does not.

The following proposition shows that prefix monotonicity and prefix-continuity are equivalent:

Proposition 3.1 (Equivalent characterizations of prefix monotonicity)**.** *Let $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ be a function. The following are equivalent: (i) f is prefix monotone (ii) $f(\langle \boldsymbol{x} \rangle) \subseteq \langle f(\boldsymbol{x}) \rangle$ for all $\boldsymbol{x} \in \mathcal{X}^*$ (iii) $\mathcal{P}(f(\boldsymbol{x})) = \mathcal{C}(f(\boldsymbol{x}))$ for all $\boldsymbol{x} \in \mathcal{X}^*$ (iv) f is prefix-continuous.*

Proof. $(i) \Rightarrow (ii)$: Prefix monotonicity means that for any $\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}^*$, we have $\boldsymbol{x} \preceq \boldsymbol{x}\boldsymbol{x}'$, so $f(\boldsymbol{x}) \preceq f(\boldsymbol{x}\boldsymbol{x}')$ and thus that there exists a $\boldsymbol{y} \in \mathcal{Y}^*$ such that $f(\boldsymbol{x}\boldsymbol{x}') = f(\boldsymbol{x})\boldsymbol{y}$. And since $\boldsymbol{x}'$ was chosen arbitrarily, (ii) holds.

$(ii) \Rightarrow (iii)$: Let $\boldsymbol{x} \in \mathcal{X}^*$. Suppose $f(\langle \boldsymbol{x} \rangle) \subseteq \langle f(\boldsymbol{x}) \rangle$. Then, $\langle \boldsymbol{x} \rangle \subseteq f^{-1}(f(\langle \boldsymbol{x} \rangle)) \subseteq \mathcal{P}(f(\boldsymbol{x}))$. Note that, for any $\boldsymbol{x}' \in \mathcal{P}(f(\boldsymbol{x}))$, $\mathcal{P}(f(\boldsymbol{x}')) \subseteq \mathcal{P}(f(\boldsymbol{x}))$. Therefore, $\langle \boldsymbol{x}' \rangle \subseteq \mathcal{P}(f(\boldsymbol{x}')) \subseteq \mathcal{P}(f(\boldsymbol{x}))$ and so $\boldsymbol{x}' \in \mathcal{C}(f(\boldsymbol{x}))$. This shows that $\mathcal{P}(f(\boldsymbol{x})) = \mathcal{C}(f(\boldsymbol{x}))$.

$(iii) \Rightarrow (iv)$: Let $\boldsymbol{y} \in \mathcal{Y}^*$ and $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$. Then $\boldsymbol{y} \preceq f(\boldsymbol{x})$, while $\boldsymbol{x} \in \mathcal{P}(f(\boldsymbol{x})) = \mathcal{C}(f(\boldsymbol{x}))$ by (iii). Hence $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(f(\boldsymbol{x})) \subseteq \mathcal{P}(\boldsymbol{y})$, so $\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y})$. As $\boldsymbol{x}$ was arbitrary, $\mathcal{P}(\boldsymbol{y}) = \mathcal{C}(\boldsymbol{y})$ and is cylindrical.

$(iv) \Rightarrow (i)$: Assume f is prefix-continuous, i.e., $\mathcal{P}(\boldsymbol{y})$ is cylindrical

for all $\boldsymbol{y}$. Let $\boldsymbol{x} \preceq \boldsymbol{x}'$. Since $f(\boldsymbol{x}) \preceq f(\boldsymbol{x})$, we have $\boldsymbol{x} \in \mathcal{P}(f(\boldsymbol{x}))$. By prefix-continuity, $\mathcal{P}(f(\boldsymbol{x}))$ is cylindrical, so $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(f(\boldsymbol{x}))$. In particular, $\boldsymbol{x}' \in \mathcal{P}(f(\boldsymbol{x}))$, which gives $f(\boldsymbol{x}) \preceq f(\boldsymbol{x}')$. ■

A strict-prefix monotone map is in particular prefix monotone, so by proposition 3.1 its precover is cylindrical and the remainder $\mathcal{R}(\boldsymbol{y})$ is empty. The next proposition, due to Vieira et al. (2025a), adds that the quotient is finite, with a size bound exponential in $|\boldsymbol{y}|$, so $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$ is computable:

Proposition 3.2 (Strict-prefix monotonicity and finite quotients)**.** *Let $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ be a strict-prefix monotone map. Then, for every $\boldsymbol{y} \in \mathcal{Y}^*$:*
1. $f^{-1}(\boldsymbol{y}) = \mathcal{Q}(\boldsymbol{y}) \setminus \bigsqcup_{y \in \mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y)$
2. $\bigsqcup_{y \in \mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y) \subseteq \mathcal{Q}(\boldsymbol{y})\,(\mathcal{X} \sqcup \{\varepsilon\})$
3. $|\mathcal{Q}(\boldsymbol{y})| \leq (|\mathcal{X}| + 1)^{|\boldsymbol{y}|}$

The following self-contained proof adapts their argument to the notation of this chapter.

Proof. (1) Observe that (using proposition 3.1) for every $\boldsymbol{y} \in \mathcal{Y}^*$

$$\mathcal{C}(\boldsymbol{y}) = \mathcal{P}(\boldsymbol{y}) = f^{-1}(\boldsymbol{y}) \sqcup \bigsqcup_{y \in \mathcal{Y}} \mathcal{P}(\boldsymbol{y}y) = f^{-1}(\boldsymbol{y}) \sqcup \bigsqcup_{y \in \mathcal{Y}} \mathcal{C}(\boldsymbol{y}y) \tag{3.9}$$

In particular, the minimality of $\mathcal{Q}(\boldsymbol{y})$ in $\mathcal{P}(\boldsymbol{y})$ implies that

$$\mathcal{Q}(\boldsymbol{y}) \cap \bigsqcup_{y \in \mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y) = \mathcal{Q}(\boldsymbol{y}) \cap \bigsqcup_{y \in \mathcal{Y}} \langle \mathcal{Q}(\boldsymbol{y}y) \rangle$$

Consequently, we obtain the identity $f^{-1}(\boldsymbol{y}) = \mathcal{C}(\boldsymbol{y}) \setminus \bigsqcup_{y \in \mathcal{Y}} \mathcal{C}(\boldsymbol{y}y) = \langle \mathcal{Q}(\boldsymbol{y}) \rangle \setminus \bigsqcup_{y \in \mathcal{Y}} \langle \mathcal{Q}(\boldsymbol{y}y) \rangle$, which in turn yields the inclusion

$$\mathcal{Q}(\boldsymbol{y}) \setminus \bigsqcup_{y \in \mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y) = \mathcal{Q}(\boldsymbol{y}) \setminus \bigsqcup_{y \in \mathcal{Y}} \langle \mathcal{Q}(\boldsymbol{y}y) \rangle \subseteq f^{-1}(\boldsymbol{y})$$

For the reverse inclusion, let $\boldsymbol{x}' \in f^{-1}(\boldsymbol{y})$. Then there exists a unique $\boldsymbol{x}_q \in \mathcal{Q}(\boldsymbol{y})$ such that $\boldsymbol{x}_q \preceq \boldsymbol{x}'$, by definition of the quotient set. Since f is strict-prefix monotone and $f(\boldsymbol{x}_q) \preceq f(\boldsymbol{x}') = \boldsymbol{y}$, we must have $\boldsymbol{x}_q = \boldsymbol{x}'$. Moreover, $\boldsymbol{x}' \notin \mathcal{Q}(\boldsymbol{y}y)$ for every $y \in \mathcal{Y}$, since otherwise $\boldsymbol{y}y \preceq f(\boldsymbol{x}') = \boldsymbol{y}$, which is impossible. This shows that $f^{-1}(\boldsymbol{y}) \subseteq \mathcal{Q}(\boldsymbol{y}) \setminus \bigsqcup_{y \in \mathcal{Y}} \mathcal{Q}(\boldsymbol{y}y)$, and completes the proof of (1).

(2) Fix $y \in \mathcal{Y}$, $\boldsymbol{y} \in \mathcal{Y}^*$ and let $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}y)$. There exists a unique $\boldsymbol{x}_y \in \mathcal{Q}(\boldsymbol{y})$ such that $\boldsymbol{x}_y \preceq \boldsymbol{x}$. Intersecting $\langle \boldsymbol{x}_y \rangle$ with both sides of (3.9), one obtains

$$\begin{aligned}\langle \boldsymbol{x}_y \rangle &= (\langle \boldsymbol{x}_y \rangle \cap f^{-1}(\boldsymbol{y})) \sqcup \bigsqcup_{y' \in \mathcal{Y}} (\langle \boldsymbol{x}_y \rangle \cap \mathcal{C}(\boldsymbol{y}y')) \\ &= (\langle \boldsymbol{x}_y \rangle \cap f^{-1}(\boldsymbol{y})) \sqcup \langle \boldsymbol{x} \rangle \sqcup C_{\boldsymbol{x}}\end{aligned}$$

where $C_{\boldsymbol{x}} \stackrel{\text{def}}{=} \bigsqcup_{y' \in \mathcal{Y}} (\langle \boldsymbol{x}_y \rangle \cap \mathcal{C}(\boldsymbol{y}y')) \setminus \langle \boldsymbol{x} \rangle$. The union before the set difference defining $C_{\boldsymbol{x}}$ is cylindrical, since it is a union of intersections of cylindrical sets. It contains no proper prefix $\boldsymbol{x}' \prec \boldsymbol{x}$. Indeed, if $\boldsymbol{x}' \in \mathcal{C}(\boldsymbol{y}y')$ for some y', then cylindricity also puts $\boldsymbol{x}$ in that set. Since $\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y}y)$ and the child precovers in (3.9) are disjoint, this forces $y' = y$. But then an element of $\mathcal{Q}(\boldsymbol{y}y)$ is a prefix of $\boldsymbol{x}' \prec \boldsymbol{x}$, contradicting the prefix-freeness of $\mathcal{Q}(\boldsymbol{y}y)$. Consequently $C_{\boldsymbol{x}}$ is cylindrical, since an extension of one of its elements could leave it only by entering $\langle \boldsymbol{x} \rangle$, which would make that element a proper prefix of $\boldsymbol{x}$. By strict-prefix monotonicity, we have $|\langle \boldsymbol{x}_y \rangle \cap f^{-1}(\boldsymbol{y})| \leq 1$. Hence either: (i) $\langle \boldsymbol{x}_y \rangle \cap f^{-1}(\boldsymbol{y}) = \emptyset$, in which case $\boldsymbol{x}_y \notin C_{\boldsymbol{x}}$, since otherwise cylindricity and $\boldsymbol{x}_y \preceq \boldsymbol{x}$ would imply $\boldsymbol{x} \in C_{\boldsymbol{x}}$, contrary to its definition. The displayed decomposition therefore gives $\boldsymbol{x} \preceq \boldsymbol{x}_y$, and hence

$$\boldsymbol{x} = \boldsymbol{x}_y \in \mathcal{Q}(\boldsymbol{y}) \cap \mathcal{Q}(\boldsymbol{y}y),$$

or (ii) $\langle \boldsymbol{x}_y \rangle \cap f^{-1}(\boldsymbol{y}) = \{\boldsymbol{x}_y\}$. Then there exists a string $z \in \mathcal{X}^*$ such that $\boldsymbol{x} = \boldsymbol{x}_y z$. The string z is nonempty, since $z = \varepsilon$ would make $\boldsymbol{x} = \boldsymbol{x}_y \in f^{-1}(\boldsymbol{y})$, contradicting $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}y) \subseteq \mathcal{P}(\boldsymbol{y}y)$. Let $z' \prec z$. If $z' \neq \varepsilon$, then $\boldsymbol{x}_y z' \in C_{\boldsymbol{x}}$. Since $C_{\boldsymbol{x}}$ is cylindrical, it contains $\langle \boldsymbol{x}_y z' \rangle$ and in particular it must contain $\langle \boldsymbol{x} \rangle$ contradicting the definition of $C_{\boldsymbol{x}}$. Therefore no non-empty proper prefix $z' \prec z$ exists, so z is a single symbol $x \in \mathcal{X}$ and

$$\boldsymbol{x} = \boldsymbol{x}_y x \in \mathcal{Q}(\boldsymbol{y})\mathcal{X}.$$

(3) For any $\boldsymbol{y} \in \mathcal{Y}^*$ and any $y \in \mathcal{Y}$, we have from (2),

$$|\mathcal{Q}(\boldsymbol{y}y)| \leq (|\mathcal{X}| + 1)|\mathcal{Q}(\boldsymbol{y})|$$

Iterating this bound along any string $\boldsymbol{y} \in \mathcal{Y}^*$, we have:

$$|\mathcal{Q}(\boldsymbol{y})| \leq (|\mathcal{X}| + 1)^{|\boldsymbol{y}|} |\mathcal{Q}(\varepsilon)| = (|\mathcal{X}| + 1)^{|\boldsymbol{y}|} \qquad \blacksquare$$

For functions that are not strict-prefix monotone, however, the decomposition may be infinite. Proposition 3.1 shows how we generalize Vieira et al.

(2025a). First, by relaxing strict-prefix monotonicity to prefix monotonicity, we support multi-symbol lookahead in the quotient. For instance, the DNA-to-amino-acid mapping is prefix monotone and requires two-symbol lookahead before committing to an output. Second, by introducing the remainder, we can handle functions that are not prefix monotone, meaning that there can be source strings that cover the target, but not all of their extensions do (e.g., Example 3.2).

Lemma 3.1 gives sufficient conditions on a transducer that guarantee a finite decomposition for every target string, even when the underlying function is not prefix monotone. The key notion is *safety*. A state is safe if it is *IP-universal* (Section 2.4, meaning that its input projection accepts all of $\mathcal{X}^*$), has **finite closure** (i.e., $|[\![\mathsf{f}_{[s]}]\!]| < \infty$), or if all its successors are safe.

Lemma 3.1 (Sufficient conditions for finite decompositions)**.** *Let $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ be a function realized by a transducer f. The decomposition $(\mathcal{Q}(\boldsymbol{y}), \mathcal{R}(\boldsymbol{y}))$ is finite for every $\boldsymbol{y} \in \mathcal{Y}^*$ if:*

(i) No ε-output cycles: *f contains no cycle in which every arc outputs ε.*

(ii) Safety: *Every state of f is **safe**, defined inductively as the smallest set such that s is safe if: (a) s is IP-universal; (b) $|[\![\mathsf{f}_{[s]}]\!]| < \infty$ (finite closure), or (c) for all transitions $s \xrightarrow{x:y} s'$, s' is safe.*

Proof. Fix $\boldsymbol{y} \in \mathcal{Y}^*$. We show that $\mathcal{Q}(\boldsymbol{y})$ and $\mathcal{R}(\boldsymbol{y})$ are finite. Let $\Pi_{\boldsymbol{y}}$ be the set of all paths in f that emit exactly $\boldsymbol{y}$, i.e., $\Pi_{\boldsymbol{y}} \stackrel{\text{def}}{=} \{s_0 \xrightarrow{x_1:y_1} s_1 \cdots s_{N-1} \xrightarrow{x_N:y_N} s_N \mid s_0 \in \mathsf{I}, y_1 \cdots y_N = \boldsymbol{y}\}$. By condition (i), the ε-output subgraph of f is acyclic, so any sub-path that emits no output has length at most $|\mathsf{S}|$. Since the total output is $\boldsymbol{y}$, each path has bounded length, and $\Pi_{\boldsymbol{y}}$ is finite. Let $\Pi_{\succeq \boldsymbol{y}}$ be the set obtained by extending every path in $\Pi_{\boldsymbol{y}}$ along all possible transitions, stopping an extension when it first reaches an IP-universal or finite-closure state. By condition (ii), every state of f is safe, in particular the end states of paths in $\Pi_{\boldsymbol{y}}$. In the least-fixed-point construction of safety, every transition from a safe state satisfying only clause (c) leads to a state added at an earlier stage. This implies that no extension path can continue indefinitely without satisfying a base case (IP-universality or finite closure). Since a finite-state transducer has finite branching and no infinite valid extension paths, König's Lemma (König, 1927) implies that the tree of extensions is finite.[a] Thus, $\Pi_{\succeq \boldsymbol{y}}$ is a finite set.

Let $\mathsf{S}_U \stackrel{\text{def}}{=} \{s \in \mathsf{S} \colon s \text{ is IP-universal}\}$ and $\mathsf{S}_C \stackrel{\text{def}}{=} \{s \in \mathsf{S} \colon |[\![\mathsf{f}_{[s]}]\!]| < \infty\}$ be the sets of states satisfying conditions (a) and (b), respectively, and

let $\mathsf{S}_S \stackrel{\text{def}}{=} \mathsf{S}_U \cup \mathsf{S}_C$. Writing $\rho_{|\rho|}$ for the end state of ρ, we can decompose the set of extended paths disjointly based on their end state:

$$\begin{aligned}\Pi_{\succeq \boldsymbol{y}} &= \left\{\rho \in \Pi_{\succeq \boldsymbol{y}} \colon \rho_{|\rho|} \in \mathsf{S}_U\right\} \sqcup \left\{\rho \in \Pi_{\succeq \boldsymbol{y}} \colon \rho_{|\rho|} \in \mathsf{S}_C\right\} \\ &= \left\{\boldsymbol{\pi} \cdot s_{|\boldsymbol{\pi}|} \xrightarrow{\cdots} s_N \mid \boldsymbol{\pi} \in \Pi_{\boldsymbol{y}}, s_N \in \mathsf{S}_U, \{s_{|\boldsymbol{\pi}|}, \dots, s_{N-1}\} \cap \mathsf{S}_S = \emptyset\right\} \\ &\quad \sqcup \left\{\boldsymbol{\pi} \cdot s_{|\boldsymbol{\pi}|} \xrightarrow{\cdots} s_N \mid \boldsymbol{\pi} \in \Pi_{\boldsymbol{y}}, s_N \in \mathsf{S}_C, \{s_{|\boldsymbol{\pi}|}, \dots, s_{N-1}\} \cap \mathsf{S}_S = \emptyset\right\}\end{aligned}$$

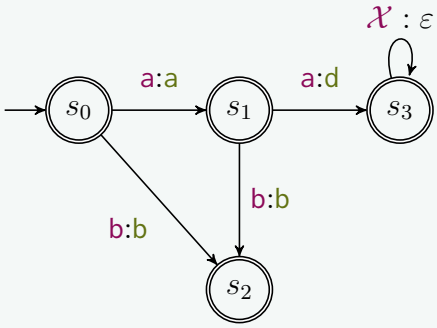
For each $\rho \in \Pi_{\succeq \boldsymbol{y}}$, let $\boldsymbol{x}_\rho$ be the source string it reads. Every $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$ is either a prefix of some $\boldsymbol{x}_\rho$, has the form $\boldsymbol{x}_\rho \boldsymbol{x}'$ where $\rho_{|\rho|} \in \mathsf{S}_C$ and $\boldsymbol{x}'$ is the source component of a pair in $[\![\mathsf{f}_{[\rho_{|\rho|}]}]\!]$, or lies in $\langle \boldsymbol{x}_\rho \rangle$ for a path ending in S_U. To see this, follow an accepting path from a root in $\Pi_{\boldsymbol{y}}$ until it ends or first reaches a base-case state. If it ends first, extend it to a stopped path. The strings of the first two forms are finite in number because there are finitely many stopped paths, each $\boldsymbol{x}_\rho$ has finitely many prefixes, and $|[\![\mathsf{f}_{[\rho_{|\rho|}]}]\!]| < \infty$ whenever $\rho_{|\rho|} \in \mathsf{S}_C$. For the third form, the path has already emitted $\boldsymbol{y}$, and functionality and IP-universality give $\langle \boldsymbol{x}_\rho \rangle \subseteq \mathcal{P}(\boldsymbol{y})$ and hence $\langle \boldsymbol{x}_\rho \rangle \subseteq \mathcal{C}(\boldsymbol{y})$. Therefore the remainder is finite. If $\boldsymbol{x}_q \in \mathcal{Q}(\boldsymbol{y})$ has the third form, then $\boldsymbol{x}_\rho \in \mathcal{C}(\boldsymbol{y})$, so some $\boldsymbol{x}'_q \in \mathcal{Q}(\boldsymbol{y})$ satisfies $\boldsymbol{x}'_q \preceq \boldsymbol{x}_\rho \preceq \boldsymbol{x}_q$. Prefix-freeness forces $\boldsymbol{x}'_q = \boldsymbol{x}_q = \boldsymbol{x}_\rho$. Such quotient elements are therefore among the finitely many stopped source strings, while all others have one of the first two forms. Hence the quotient is finite as well. ■

The transducer in Example 3.4 illustrates the safety conditions concretely.

Example 3.4 — Three ways for a state to be safe

Consider the transducer below over $\mathcal{X} = \{\mathsf{a}, \mathsf{b}\}$, $\mathcal{Y} = \{\mathsf{a}, \mathsf{b}, \mathsf{d}\}$, encoding a (partial) function f. Assigning b to every otherwise undefined input gives a total rational extension and leaves the quotient and remainder below unchanged.

[a] König's Lemma states that every infinite tree with finite branching must have an infinite path. Since our alphabet is finite (finite branching) and the Safety condition forbids cycles among non-base-case states (no infinite paths), the resulting tree of extensions must be finite.



The four states illustrate all cases of safety. State s_3 is IP-universal: it is accepting and loops on every input symbol, so its input projection accepts all of $\mathcal{X}^*$ (base case a). State s_2 has finite closure: it is accepting with no outgoing transitions, so $[\![\mathsf{f}_{[s_2]}]\!] = \{(\varepsilon, \varepsilon)\}$ is finite (base case b). State s_1 is neither IP-universal nor finite closure, but is safe by the recursive step (c): its successors s_3 (via a) and s_2 (via b) are both safe. Similarly, s_0 is safe because its successors s_1 (via a) and s_2 (via b) are both safe.

For target $\boldsymbol{y} = \mathsf{a}$, the path set $\Pi_{\mathsf{a}} = \{s_0 \xrightarrow{\mathsf{a}:\mathsf{a}} s_1\}$ is finite, with terminal state s_1 safe (condition ii). Extending from s_1: input a reaches the IP-universal state s_3, contributing aa to $\mathcal{Q}(\mathsf{a})$, and input b reaches the finite-closure state s_2, contributing ab to $\mathcal{R}(\mathsf{a})$. The root string a is also in $\mathcal{R}(\mathsf{a})$. State s_1 is accepting with $f(\mathsf{a}) = \mathsf{a} \succeq \mathsf{a}$, but $\langle \mathsf{a} \rangle \not\subseteq \mathcal{P}(\mathsf{a})$, because every strict extension of ab is assigned b by the total extension and therefore does not cover a. Thus $\mathcal{Q}(\mathsf{a}) = \{\mathsf{aa}\}$ and $\mathcal{R}(\mathsf{a}) = \{\mathsf{a}, \mathsf{ab}\}$. The ε-output loop at s_3 violates the global condition (i), although the displayed target has a finite decomposition.

The conditions in Lemma 3.1 guarantee exact computation. In particular, these are satisfied by two of the transducers used in the experiments of Chapter 4 (Section 4.4): the token-to-byte transducer f_{α} and the codon-table transducer $\mathsf{f}_{\text{dna2aa}}$, whose quotients are finite and remainders empty, but not by the PTB transducer f_{ptb}, whose quotients are infinite.

The finiteness of decomposition is a property of the function f, not of any particular transducer encoding it. The lemma's conditions are sufficient. The decomposition algorithm tests universality of whole frontiers (sets of state/string pairs), which can succeed even when no covering state is IP-universal on its own (Example 4.4). For many functions, the prefix decomposition is infinite or otherwise prohibitively large. In these cases, we let the source language model determine which prefix decomposition members carry the most significant mass and prune others (Section 4.2). This lets us

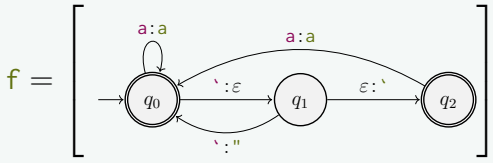
greedily approximate the probabilities of the transduced language model by enumerating a high-mass subset of the decomposition.

3.4 Example decompositions[3]

The following three small machines illustrate the sufficient conditions of Section 3.3. Two have finite decompositions, and one has an infinite remainder.

Example 3.5 — Merging consecutive backticks

The transducer below merges consecutive backticks (`` ` ` ``) into a quotation mark (`"`) but otherwise keeps the symbols `a` and `` ` `` fixed. It is a small example of text normalization.



When computing the prefix probability $\overrightarrow{p_{\mathcal{Y}}}(\texttt{`})$, sequences starting with two backticks get mapped to a quote and thus do not appear in the precover. Instead, a single backtick remains in the remainder, while a backtick followed by `a` goes into the quotient, as any further extensions preserve the initial backtick. We thus have (by visually canceling out continuations that get mapped to quotation marks),

$$\begin{aligned}\mathcal{P}(\texttt{`}) &= \{\texttt{`}, \cancel{\texttt{``}}, \cancel{\texttt{```}}, \cancel{\texttt{``a}}, \ldots\} \cup \{\texttt{`a}, \texttt{`a`}, \texttt{`aa}, \ldots\} \\ &= \{\texttt{`}\} \sqcup \{\texttt{`a}\}\,\mathcal{X}^*\end{aligned}$$

Thus, $\overrightarrow{p_{\mathcal{Y}}}(\texttt{`}) = p_{\mathcal{X}}(\texttt{`}) + \overrightarrow{p_{\mathcal{X}}}(\texttt{`a})$. Similarly, we have $\mathcal{P}(\texttt{"}) = \{\texttt{``}\}\mathcal{X}^*$ and thus $\overrightarrow{p_{\mathcal{Y}}}(\texttt{"}) = \overrightarrow{p_{\mathcal{X}}}(\texttt{``})$.

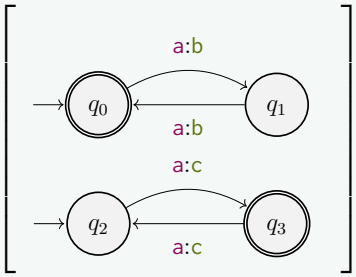
Example 3.5 demonstrates the efficiency of using the remainder and the quotient. Both sets have only a single element, yet together they determine $\overrightarrow{p_{\mathcal{Y}}}(\texttt{`})$ exactly. There are, however, edge cases that do not lend themselves to such efficient decompositions.

[3] In Snæbjarnarson et al. (2026a), this section appears as an appendix.

Example 3.6 — An infinite remainder

Consider the following mapping f and its representation as a nondeterministic transducer f with two initial states and two disjoint 2-cycles.

$$f(\mathsf{a}^n) = \mathsf{b}^n \textbf{ if } n \text{ is even } \textbf{else } \mathsf{c}^n \qquad \mathsf{f} =$$



with $\mathcal{X} = \{\mathsf{a}\}$ and $\mathcal{Y} = \{\mathsf{b}, \mathsf{c}\}$. The precover for b is given by

$$\mathcal{P}(\mathsf{b}) = \{\cancel{\varepsilon}, \cancel{\mathsf{a}}, \mathsf{aa}, \cancel{\mathsf{aaa}}, \mathsf{aaaa}, \cancel{\mathsf{aaaaa}}, \mathsf{aaaaaa}, \ldots\}$$

This example is, unfortunately, challenging for our approach because the remainder set is infinite:

$$\mathcal{R}(\mathsf{b}) = \left\{\mathsf{a}^{2^n} \mid n > 0\right\}, \mathcal{Q}(\mathsf{b}) = \emptyset$$

In other words, the mapping never releases the evenness constraint, and we sum forever.

Example 3.7 — A one-element remainder

Consider the following mapping f, visualized in Figure 3.1.

$$f(\mathsf{a}^n) = \mathsf{b}^n \textbf{ if } n \neq 2 \textbf{ else } \mathsf{c}$$

Here, $\mathcal{X} = \{\mathsf{a}\}$ and $\mathcal{Y} = \{\mathsf{b}, \mathsf{c}\}$. Suppose we want to compute the prefix probability $\overrightarrow{p_{\mathcal{Y}}}(\mathsf{b})$.

$$\mathcal{P}(\mathsf{b}) = \{\mathsf{a}, \cancel{\mathsf{aa}}, \mathsf{aaa}, \mathsf{aaaa}, \mathsf{aaaaa}, \ldots\}$$

$$\mathcal{R}(\mathsf{b}) = \{\mathsf{a}\}, \mathcal{Q}(\mathsf{b}) = \{\mathsf{aaa}\}$$

Thus,

$$\begin{aligned}\overrightarrow{p_{\mathcal{Y}}}(\mathsf{b}) &= \sum_{\boldsymbol{x} \in \mathcal{R}(\mathsf{b})} p_{\mathcal{X}}(\boldsymbol{x}) + \sum_{\boldsymbol{x} \in \mathcal{Q}(\mathsf{b})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \\ &= p_{\mathcal{X}}(\mathsf{a}) + \overrightarrow{p_{\mathcal{X}}}(\mathsf{aaa})\end{aligned}$$

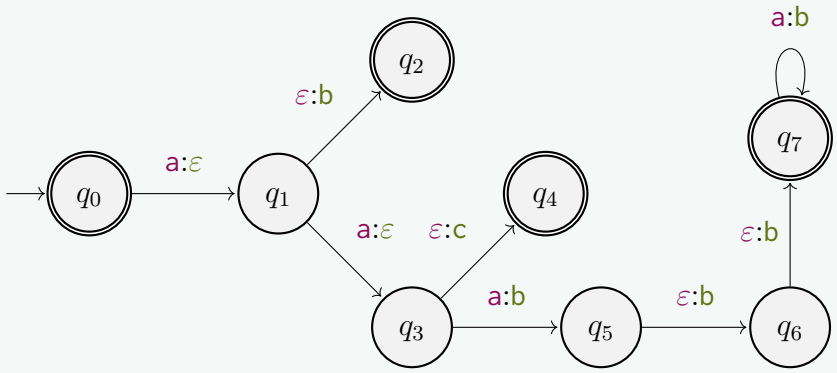


Figure 3.1: An FST that maps n occurrences of a to the same number of b's, except when the input is exactly two a's, which are mapped to c.

The backtick normalizer (Example 3.5) and the exceptional-input map (Example 3.7) satisfy both conditions of Lemma 3.1. Neither contains an ε-output cycle, and every state is safe. For the targets considered above, both machines have a one-element quotient and a one-element remainder. The evenness machine (Example 3.6) fails condition (ii). Within each of its two disjoint 2-cycles, the states are mutually reachable and none is IP-universal or has finite closure. The inductive definition therefore has no base case in either cycle, so none of those states is safe. As calculated above, $\mathcal{R}(\mathsf{b})$ is infinite.

3.5 Relation to prior constructions

The closest prior work is Vieira et al. (2025a), which converts a token-level language model into a character-level one and computes the character-level prefix probabilities exactly. Their construction is the special case of a transduced language model in which the transformation is a strict-prefix monotone map from token strings to character strings, and their finiteness guarantee corresponds to proposition 3.2. Section 3.3 extends their setting to prefix-monotone functions that may read several source symbols before producing the next target symbol and, through the remainder, to functions that are not prefix monotone. The determinized precover machine and the algorithms of Chapter 4 reduce to their tokens-to-characters algorithm under these assumptions. Closely related work obtains alternative algorithms for the same decoding map by treating only the tokenizer's canonical encoding of each string as valid. Phan et al. (2024) show that querying a language model after tokenizing a text prefix need not recover the corresponding next-character distribution, and derive corrections for maximum-prefix and BPE tokenization. Phan et al. (2025)

express byte probabilities as sums over canonically valid covers and apply the resulting interface to in-filling and model ensembles. Their exactness assumes zero probability on noncanonical encodings; for softmax language models, they zero invalid next-token probabilities and renormalize the rest. For BPE, Hayase et al. (2026) show that the canonically valid paths covering a byte prefix share a fixed token prefix and differ only within a bounded suffix, which their algorithm maintains and marginalizes incrementally. These methods improve inference for the same strict-prefix-monotone map under stronger support assumptions—they do not broaden the class of transformations to include output-delaying, rewriting, or deleting maps.

3.5.1 The tokens-to-characters special case

Vieira et al. (2025a) study the case in which the source strings are subword tokens and the transformation decodes token strings to the character strings they spell. Their tokenization model is a tuple of a character alphabet, a token alphabet, a stochastic encoder, and a decoder κ from token strings to character strings. The decoder is this chapter's f with the alphabets in their roles here: tokens are the source alphabet and characters the target. Their algorithms assume κ is strict-prefix monotone, an assumption the BPE and WordPiece decoders satisfy because every token spells at least one character. Their *prefix-encodings* of a character prefix, the token strings whose decodings extend it, are the precover $\mathcal{P}(\boldsymbol{y})$. Their *covering* consists of the shortest token prefixes that cover the character prefix. By proposition 3.1, strict-prefix monotonicity makes the precover the union of the cylinders generated by these shortest prefixes. The covering is therefore the quotient $\mathcal{Q}(\boldsymbol{y})$, and the remainder is empty.

Their algorithm. The algorithm of Vieira et al. (2025a) maintains a set of covering token prefixes, each carrying the token string, its prefix probability under the source model, and its decoding. To extend a target prefix by one character, the algorithm starts from the covering of the previous target prefix. A covering token prefix whose decoding already extends beyond the previous target prefix is kept unchanged if its decoding also covers the extended target prefix. If its decoding ends exactly at the previous target prefix, the algorithm appends every possible token and keeps the children whose decodings cover the extended target prefix. An unchanged item retains its prefix probability. A child's prefix probability is its parent's prefix probability multiplied by the model's next-token conditional. The retained items form the covering of the extended target prefix. The next-character conditional probability is the ratio of the summed prefix probabilities of the new and previous coverings. Run exhaustively, the loop is exact (their Prop. 1). The covering can grow

exponentially in the target length, so their *beam-summing* approximation groups covering prefixes that differ only in an overshooting final token and retains the K highest-mass groups.

Algorithmic correspondence. Taking the character alphabet in their construction to be the byte alphabet, consider the token-to-byte transducer $\mathtt{f}_\alpha$. Every state is IP-universal (Table A.4), and every non-ε-input arc emits output. Under these conditions, a source prefix belongs to $\mathcal{C}(\boldsymbol{y})$ exactly when it belongs to $\mathcal{P}(\boldsymbol{y})$, since every covering frontier state is IP-universal (Proposition 4.3). The remainder is therefore empty at every target. Because every non-ε-input arc emits output, `decompose` classifies each newly discovered prefix immediately. This is the all-universal fast path described in Section 4.3.9. The resulting fast path matches their loop for advancing a set of covering token prefixes byte by byte under the source model's conditionals. The probability-mass prune of Section 4.3.4 plays a role analogous to their beam width, under a different retention rule.

Beyond strict-prefix-monotone decoding. Three features supported by transduced language models fall outside their strict-prefix-monotone decoding setting. For the DNA-to-amino-acid map (Example 3.3), zero-output source symbols require the frontier to continue without advancing the target. For the Newspeak transducer (Example 3.2), some covering prefixes have extensions outside the precover, producing a nonempty remainder. For the evenness machine (Example 3.6), the decomposition can be infinite, so exhaustive enumeration may not terminate.

Chapter 4

Inference

This chapter contains material reproduced and adapted from Snæbjarnarson et al. (2026a), with editorial changes for narrative consistency.

Chapter 3 reduced the prefix probability of a transduced language model to a sum over the precover of the target prefix, in the form of the quotient–remainder decomposition. This chapter develops the algorithms to compute that decomposition and, with them, the full autoregressive interface.

4.1 The autoregressive interface[1]

We seek algorithms for the prefix probabilities $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$, string probabilities $p_{\mathcal{Y}}(\boldsymbol{y})$, and next-symbol distributions $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$ of transduced language models. Together these constitute the *autoregressive interface* to the transduced language model (Section 2.2). The rest of the chapter develops an abstract enumerative algorithm (Section 4.2) and an efficient, frontier-based realization (Section 4.3). Both are exact without pruning and reduce the interface to a single primitive, the prefix decomposition `decompose`$(\boldsymbol{y}) = (\mathtt{Q}, \mathtt{R})$ of Equation (3.7): The primitive operation is `prefix_prob` (Algorithm 4.1), which requires a single call to `decompose`. Both `next_dist` and `prob` are derived from it: `next_dist` computes $|\mathcal{Y}|$ additional prefix probabilities and obtains $\overrightarrow{p_{\mathcal{Y}}}(\text{EOS} \mid \boldsymbol{y})$ by complement, and `prob` is a one-line product. In an implementation, $\overrightarrow{p_{\mathcal{X}}}$ and $p_{\mathcal{X}}$ should be memoized so that extending $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ to $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}x)$ requires only a single conditional evaluation rather than replaying the entire history (see Section 4.3.7). In Section 4.3.5, we show how to compute `next_dist` with a more efficient, joint decomposition, rather than $|\mathcal{Y}|+1$ separate calls to `prefix_prob`.

[1] In Snæbjarnarson et al. (2026a), this material appears as part of an appendix.

```
def prefix_prob(y):
  (Q, R) ← decompose(y)
  return Σ_{x∈Q} p→_X(x) + Σ_{x∈R} p_X(x)

def prob(y):
  return prefix_prob(y) · next_dist(y)[EOS]

def next_dist(y):
  p̄ ← {}
  Z ← prefix_prob(y)
  if Z = 0: return {EOS : 1}
  for y' ∈ 𝒴:
    p̄[y'] ← prefix_prob(yy')/Z
  p̄[EOS] ← 1 − Σ_{y'∈𝒴} p̄[y']
  return p̄
```

Algorithm 4.1: Interface to the transduced language model. `prefix_prob` is the primitive, and `next_dist` and `prob` derive from it.

The three checks invoked by `decompose` (`is_cylinder`, `is_member`, and `is_live`, Algorithm 4.2) all require determining which transducer states are reachable after reading a source string $\boldsymbol{x}$, and what output each has produced relative to $\boldsymbol{y}$. The efficient algorithms of Section 4.3 capture this information in a single data structure, the *frontier*, and reduce each check to a frontier query. The quotient and remainder may be too large to enumerate exhaustively, so we introduce pruning (Section 4.3.4) to obtain practical approximations.

4.2 Exact inference

We now present an algorithm for computing prefix decompositions by exploiting the explicit structure of a transducer that encodes the function. Combined with the computational shortcut (Equation (3.6)), this gives an autoregressive interface to transduced language models.

Algorithm 4.2 gives the decomposition algorithm (`decompose`), which maintains a queue of candidate source strings and explores them by breadth-first search (BFS), optionally pruning low-probability candidates at each step (described below).[2] Each dequeued string $\boldsymbol{x}$ undergoes three checks, defined

[2]For efficiency, `decompose` should be memoized. For $|\boldsymbol{y}| > 0$, the recursive call `decompose`($\boldsymbol{y}_{<N}$) seeds the queue with the previous decomposition $\mathsf{Q}' \cup \mathsf{R}'$ rather than

```
def decompose(y): # memoize
  N ← |y|
  if N = 0:
    P ← QUEUE({ε})
  else:
    (Q′, R′) ← decompose(y_<N)
    P ← QUEUE(Q′ ∪ R′)
  (Q, R) ← (∅, ∅)
  while |P| > 0:
    P′ ← ∅
    for x ∈ P:
      if is_cylinder(x, y):
        Q.add(x)
        continue
      if is_member(x, y):
        R.add(x)
      for x′ ∈ X:
        if is_live(xx′, y):
          P′.add(xx′)
    P ← prune(P′)
  return (Q, R)
```

Algorithm 4.2: The `decompose` algorithm enumerates the quotient and remainder of a target prefix using the `is_cylinder`, `is_member`, and `is_live` checks instantiated in Section 4.2.1.

in terms of the precover $\mathcal{P}(\boldsymbol{y})$:

1. *Cylinder*: $\texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y}) \iff \langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$, i.e., every extension of $\boldsymbol{x}$ covers $\boldsymbol{y}$. $\boldsymbol{x}$ is added to the quotient set `Q` and not explored further (Section 4.2).
2. *Member*: $\texttt{is_member}(\boldsymbol{x}, \boldsymbol{y}) \iff \boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$, i.e., $\boldsymbol{x}$ itself covers $\boldsymbol{y}$. When `is_cylinder` is false but `is_member` is true, $\boldsymbol{x}$ is added to the remainder set `R`, and its extensions are still explored (Section 4.2).
3. *Live*: $\texttt{is_live}(\boldsymbol{x}, \boldsymbol{y}) \iff \exists \boldsymbol{x}'' \in \mathcal{X}^* : \boldsymbol{x}\boldsymbol{x}'' \in \mathcal{P}(\boldsymbol{y})$, i.e., some extension of $\boldsymbol{x}$ belongs to the precover. Only live extensions are enqueued (Section 4.2).

Because strings are processed shortest-first, if any prefix of the current string had already entered the quotient, the current string would never have been enqueued. Once the queue is exhausted, the algorithm returns the prefix

enumerating from ε.

decomposition. We instantiate these checks concretely in Section 4.2.1.

Example 4.1 — Tracing `decompose` on the lowercasing transducer

We trace a nonrecursive breadth-first search from ε that is equivalent to Algorithm 4.2 on the lowercasing transducer of Example 3.1 with target $\boldsymbol{y} = \mathtt{ab}$, restricting the source alphabet to $\{\mathtt{A}, \mathtt{a}, \mathtt{B}, \mathtt{b}\}$ for readability. Each row dequeues one source string $\boldsymbol{x}$ and applies the three checks against $\mathcal{P}(\mathtt{ab})$:

$\boldsymbol{x}$	$f(\boldsymbol{x})$	`is_cylinder`	`is_member`	live children	action
ε	ε	✗ ($\mathtt{B} \mapsto \mathtt{b}$)	✗	A, a	extend
A	a	✗ ($\mathtt{AA} \mapsto \mathtt{aa}$)	✗	AB, Ab	extend
a	a	✗	✗	aB, ab	extend
AB, Ab, aB, ab	ab	✓	—	none	$\to$ Q

In round one, B and b are never enqueued: `is_live` fails because every extension lowercases to a string starting with $\mathtt{b} \neq \mathtt{a}$. In round two, the cylinder check fails for A because the extension AA maps to aa, which diverges from the target, so A must be extended rather than absorbed. In round three every dequeued string maps exactly to ab and every extension keeps that prefix, so all four enter the quotient and, having no live work left, the queue empties. The result $\mathtt{Q} = \{\mathtt{AB}, \mathtt{Ab}, \mathtt{aB}, \mathtt{ab}\}$, $\mathtt{R} = \emptyset$ is exactly the decomposition summed by hand in Equation (3.4c). On Example 3.2 with $\boldsymbol{y} = \mathtt{ba}$ the same loop instead puts ba in the *remainder*: it passes `is_member` but fails `is_cylinder` because its child bad dies, reproducing Equation (3.5a).

Theorem 4.1 (Correctness of `decompose`)**.** *If $\mathcal{P}(\boldsymbol{y}')$ admits a finite decomposition for every prefix $\boldsymbol{y}' \preceq \boldsymbol{y}$ and the three checks exactly implement the conditions above with no pruning, then* `decompose`$(\boldsymbol{y})$ *terminates and its output* $(\mathtt{Q}, \mathtt{R})$ *is the optimal prefix decomposition (Equation* (3.7)*).*

Proof. We first prove the claim for the nonrecursive form of `decompose`, which initializes the queue with $\{\varepsilon\}$ for every target.

We verify the three conditions of the optimal prefix decomposition (Equation (3.7)).

1. *Prefix-freeness* (Q is prefix-free): When `is_cylinder` succeeds for $\boldsymbol{x}$

(Section 4.2), the `continue` (Section 4.2) skips the extension loop, so no $\boldsymbol{x}x'$ is enqueued. Since strings enter the queue only as single-symbol extensions of dequeued strings, no extension of $\boldsymbol{x}$ is ever enqueued or dequeued, and thus no two elements of $\mathtt{Q}$ share a prefix.

2. *Validity* ($\mathcal{P}(\boldsymbol{y}) = \langle \mathtt{Q} \rangle \sqcup \mathtt{R}$): Let $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$. We first show that $\boldsymbol{x} \in \langle \mathtt{Q} \rangle \sqcup \mathtt{R}$. We show by induction on prefix length that either a prefix of $\boldsymbol{x}$ enters $\mathtt{Q}$ or $\boldsymbol{x}$ itself is dequeued and added to $\mathtt{Q}$ or $\mathtt{R}$. The base case holds: ε is enqueued. If $\boldsymbol{x}_{<k}$ is dequeued and not placed in $\mathtt{Q}$, then $\boldsymbol{x}_{<k+1}$ is enqueued by the extension loop: since $\boldsymbol{x}_{<k+1} \preceq \boldsymbol{x}$ and $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$, we have $\mathtt{is_live}(\boldsymbol{x}_{<k+1}, \boldsymbol{y})$ (Section 4.2). By induction, either some prefix $\boldsymbol{x}_{<j}$ enters $\mathtt{Q}$ (giving $\boldsymbol{x} \in \langle \boldsymbol{x}_{<j} \rangle \subseteq \langle \mathtt{Q} \rangle$) or $\boldsymbol{x}$ itself is dequeued, the cylinder check fails, and $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$ gives $\boldsymbol{x} \in \mathtt{R}$ via `is_member` (Section 4.2). The reverse direction is clear since elements of $\mathtt{Q}$ pass the exact `is_cylinder` check, so $\langle \mathtt{Q} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$. Similarly, elements of $\mathtt{R}$ pass `is_member`, so $\mathtt{R} \subseteq \mathcal{P}(\boldsymbol{y})$. What remains is to show that $\langle \mathtt{Q} \rangle$ and $\mathtt{R}$ are disjoint. If $\boldsymbol{x} \in \mathtt{R}$, then $\boldsymbol{x}$ was dequeued and `is_cylinder` failed, so $\boldsymbol{x} \notin \mathtt{Q}$. No proper prefix of $\boldsymbol{x}$ is in $\mathtt{Q}$ either: otherwise $\boldsymbol{x}$ would never have been enqueued (prefix-freeness). Hence $\boldsymbol{x} \notin \langle \mathtt{Q} \rangle$.
3. *Maximality* ($\langle \mathtt{Q} \rangle = \mathcal{C}(\boldsymbol{y})$): shortest-first processing ensures no proper prefix of a quotient element satisfies `is_cylinder` (Section 4.2), so $\mathtt{Q}$ is the *minimal* prefix-free set of cylinders. Together with the exact cylinder check, the earliest qualifying prefix is always found, as required by Equation (3.7). It remains to show $\mathcal{C}(\boldsymbol{y}) \subseteq \langle \mathtt{Q} \rangle$. By validity, $\mathcal{P}(\boldsymbol{y}) = \langle \mathtt{Q} \rangle \sqcup \mathtt{R}$. Every element of $\mathtt{R}$ fails `is_cylinder`, so $\mathtt{R} \cap \mathcal{C}(\boldsymbol{y}) = \emptyset$. Hence $\mathcal{C}(\boldsymbol{y}) \subseteq \langle \mathtt{Q} \rangle$.

Termination: `is_live` (Section 4.2) ensures only live extensions are enqueued, and finiteness of the decomposition guarantees the queue empties.

For the recursive implementation, write $N = |\boldsymbol{y}|$. The case $N = 0$ is the nonrecursive form. For $N > 0$, assume by induction that $\mathtt{decompose}(\boldsymbol{y}_{<N})$ returns its optimal decomposition $(\mathtt{Q}', \mathtt{R}')$. Since $\mathcal{P}(\boldsymbol{y}) \subseteq \langle \mathtt{Q}' \rangle \sqcup \mathtt{R}'$, every relevant string either belongs to the initial queue or extends one of its elements. No element of $\mathtt{R}'$, and no proper prefix of an initial element, can be a cylinder for $\boldsymbol{y}$. Replacing $\{\varepsilon\}$ with $\mathtt{Q}' \cup \mathtt{R}'$ therefore preserves the resulting decomposition. The recursive calls terminate under the theorem's prefix-finiteness hypothesis. ■

Conservative checks and suboptimality. If the checks used by `decompose` are conservative approximations, the algorithm may produce *suboptimal* decompositions: valid but with a smaller quotient than the optimal one. A conservative liveness check (false positives) enqueues unnecessary extensions

but does not affect classification: every dequeued string is still correctly classified by `is_cylinder` and `is_member`, so if `decompose` terminates, its result remains optimal. A conservative cylinder check (false negatives) may fail to recognize some quotient elements, placing them in `R` instead. The decomposition remains valid, but because a successful cylinder check terminates exploration of that subtree (Section 4.2), missing a cylinder means the BFS continues exploring extensions that would otherwise have been cut off, potentially inflating the remainder and increasing the computation. By contrast, the membership check must be exact to preserve validity.

Approximation via pruning. When the prefix decomposition grows large, exhaustive enumeration and scoring may become infeasible. In these cases, `prune` retains the highest-probability candidates until their cumulative mass reaches $(1 - \tau)$ of the candidate mass or their count reaches n_{max}, whichever comes first. This discards low-probability candidates and reduces the computation. Since pruning only removes candidates from the queue, every element found is correct ($\langle \mathtt{Q} \rangle \sqcup \mathtt{R} \subseteq \mathcal{P}(\boldsymbol{y})$), but the decomposition is no longer valid in general (coverage may be incomplete), so the computed prefix probability is a lower bound on the true value. Our strategy is detailed in Section 4.3.4.

4.2.1 Getting started: instantiating the checks with the precover machine

We now show how to instantiate the three checks (Algorithm 4.2) using a finite-state transducer. The faster algorithms used in practice follow in Section 4.3.

We represent the transformation f with a transducer `f` (see Section 2.4). Given a target prefix $\boldsymbol{y}$, $\mathsf{proj}_{\mathcal{X}}(\mathtt{f} \circ \boldsymbol{y}\mathcal{Y}^*)$ is an NFA that accepts exactly $\mathcal{P}(\boldsymbol{y})$. To enable the efficient state-based checks below, we determinize and trim this NFA to obtain the DFA

$$\mathtt{P}_{\boldsymbol{y}} \stackrel{\text{def}}{=} \mathsf{trim}(\mathsf{determinize}(\mathsf{proj}_{\mathcal{X}}(\mathtt{f} \circ \boldsymbol{y}\mathcal{Y}^*))).$$

Let $\mathsf{S}_{\boldsymbol{y}}$, $\mathsf{I}_{\boldsymbol{y}}$, $\mathsf{F}_{\boldsymbol{y}}$, and $\mathsf{T}_{\boldsymbol{y}}$ denote the components of $\mathtt{P}_{\boldsymbol{y}}$. Since $\mathtt{P}_{\boldsymbol{y}}$ is deterministic, scanning a string $\boldsymbol{x} = x_1 \cdots x_N$ gives a unique state, returned by $\mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$.

Example 4.2 — Building the precover machine

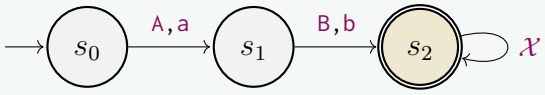
Build $\mathtt{P}_{\mathsf{ab}}$ for the lowercasing transducer `f` of Example 3.1, with the source alphabet restricted to $\{\mathtt{A}, \mathtt{a}, \mathtt{B}, \mathtt{b}\}$ as in Example 4.1.

1. *Compose.* The copy-transducer for the cylinder $\mathtt{ab}\mathcal{Y}^*$ is the chain $c_0 \xrightarrow{\mathtt{a}}$

$c_1 \xrightarrow{\mathtt{b}} c_2$ with c_2 accepting and looping on every symbol. Composition runs f and the chain in lockstep, the chain consuming f's *output*: from (q_0, c_0) the arcs A:a and a:a advance the chain to (q_0, c_1), while B:b and b:b die (c_0 has no b-arc). From (q_0, c_1) only the b-emitting arcs survive, reaching (q_0, c_2). From (q_0, c_2) every arc loops. Only (q_0, c_2) is accepting, since both components must accept.

2. *Project.* Dropping the output side leaves an automaton over the source alphabet that, by construction, accepts exactly $\mathcal{P}(\mathtt{ab})$.
3. *Determinize.* This projection happens to be input-deterministic already (one arc per input symbol from each state), so the subset construction has nothing to merge.
4. *Trim.* Every state lies on an accepting path, so nothing is removed either.

The result is the three-state DFA



whose language is $\{\mathtt{A}, \mathtt{a}\}\{\mathtt{B}, \mathtt{b}\}\mathcal{X}^*$, the four cylinders of Example 3.1 in machine form. This machine makes the checks in Example 4.1 explicit: $\mathtt{run}_{\mathtt{ab}}(\mathtt{A}) = s_1$, and the cylinder check for AB succeeds because s_2 (gold) is universal.

For the Newspeak transducer, composition restricts the output to $\mathtt{ba}\mathcal{Y}^*$. After projection, multiple runs may remain for the same source string, so determinization combines them. Trimming removes any states with no accepting continuation. The resulting DFA recognizes $\mathcal{P}(\mathtt{ba})$. A minimized equivalent DFA is the four-state machine drawn after Equation (3.6), with its teal remainder state and gold universal state.

We now describe how to implement the three checks using $\mathtt{P}_{\boldsymbol{y}}$ (pseudocode in Algorithm 4.3).

- *Cylinder*: We need to check whether $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$, equivalently whether $\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y})$. The algorithm adds the prefix-minimal strings passing this check, so they form $\mathcal{Q}(\boldsymbol{y}) = \mathrm{pf}(\mathcal{C}(\boldsymbol{y}))$. Let $S = \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$ be the unique state reached after scanning $\boldsymbol{x}$. Since $\mathtt{P}_{\boldsymbol{y}}$ is deterministic, $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$ if and only if S is **universal**: $[\![\mathtt{P}_{\boldsymbol{y}[S]}]\!] = \mathcal{X}^*$, meaning every continuation of $\boldsymbol{x}$ is accepted. The `is_cylinder` check (Algorithm 4.3) tests universality via breadth-first search (BFS) from S.
- *Member*: When the `is_cylinder` check fails for $\boldsymbol{x}$ with $S = \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$, we need to determine whether the scanned string $\boldsymbol{x}$ is in $\mathcal{R}(\boldsymbol{y})$. Here it suffices

to check if $S \in \mathsf{F}_{\boldsymbol{y}}$, i.e., if $\mathtt{P}_{\boldsymbol{y}}$ accepts $\boldsymbol{x}$.

- *Live*: We construct $\mathtt{P}_{\boldsymbol{y}}$ as a *trimmed* automaton accepting $\boldsymbol{x}$ where $\boldsymbol{y} \preceq f(\boldsymbol{x})$.[3] Trimming ensures that every reachable state lies on some accepting path, so liveness reduces to $\mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x}) \neq \emptyset$.

```
def step_y(s, x):
  return T_y(s, x) # deterministic

def run_y(x_1 ··· x_N):
  if N = 0: return I_y # unique
  return step_y(run_y(x_{<N}), x_N)

def is_member(x, y):
  return run_y(x) ∈ F_y

def is_live(x, y):
  # not in failure state
  return run_y(x) ≠ ∅

def is_cylinder(x, y):
  # Run BFS to find a counterexample:
  # a nonaccepting or incomplete state
  S ← run_y(x); V ← {S}; 𝒫 ← QUEUE({S})
  while 𝒫:
    S ← 𝒫.pop()
    if S ∉ F_y: return False
    for x' ∈ 𝒳:
      S' ← step_y(S, x')
      if S' = ∅: return False
      if S' ∉ V: 𝒫.add(S'); V.add(S')
  return True   # No counterexamples
```

Algorithm 4.3: State-based instantiation of the checks from Algorithm 4.2 on the precover DFA $\mathtt{P}_{\boldsymbol{y}}$. Helper functions `step` and `run` advance through the DFA. The checks `is_member` and `is_live` are single-state lookups, and `is_cylinder` performs a BFS to verify universality.

Although `decompose` is written in terms of source strings, each check internally computes the state $S = \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$ reached by scanning $\boldsymbol{x}$ in the deterministic $\mathtt{P}_{\boldsymbol{y}}$, and reduces to a state property. Section A.1.4 gives an

[3] This can also be expressed as $\mathcal{P}(\boldsymbol{y}) = f^{-1}(\boldsymbol{y}\mathcal{Y}^*) = [\![\mathsf{proj}_{\mathcal{X}}(\mathsf{f} \circ \boldsymbol{y}\mathcal{Y}^*)]\!]$.

alternative algorithm that traverses the states of $\mathtt{P}_{\boldsymbol{y}}$ and represents the quotient and remainder as DFAs.

4.3 Efficient algorithms[4]

The decomposition algorithm of Section 4.2 operates on a determinized, trimmed precover machine $\mathtt{P} = \mathsf{trim}(\mathsf{determinize}(\mathsf{proj}_{\mathcal{X}}(\mathtt{f} \circ \boldsymbol{y}\mathcal{Y}^*)))$. In practice, both the composition with $\boldsymbol{y}\mathcal{Y}^*$ and the explicit determinization are expensive and must be repeated for each target $\boldsymbol{y}$. The algorithms in this section perform the composition and determinization lazily, operating directly on a trimmed version of $\mathtt{f}$ and avoiding explicit materialization. This requires careful bookkeeping of partial outputs via *frontiers* (Section 4.3.1), but enables precomputation of IP-universal states and incremental reuse across targets.

We introduce the frontier data structure (Section 4.3.1) and show how each check from Algorithm 4.2 reduces to a frontier query (Section 4.3.2). The resulting algorithms realize the autoregressive interface of Section 4.1. We write $(\mathsf{S}, \mathcal{X}, \mathcal{Y}, \mathsf{I}, \mathsf{F}, \mathsf{T}) = \mathtt{f}$ throughout and omit the dependency on $\overrightarrow{p_{\mathcal{X}}}$ since it is clear from context.

4.3.1 Frontier computation

Since $\mathtt{f}$ can be nondeterministic even if it is functional, it may reach many states simultaneously for a given source string $\boldsymbol{x}$ and target $\boldsymbol{y}$. Each of these paths may have produced a different output buffer. The **frontier** $\mathcal{F} = \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$ collects this information: it is the set of $(s, \boldsymbol{b})$ pairs (where $s \in \mathsf{S}$ is a transducer state and $\boldsymbol{b} \in \mathcal{Y}^*$ is the output produced so far) reachable by reading $\boldsymbol{x}$ from the initial states, filtered to buffers $\boldsymbol{b}$ compatible with $\boldsymbol{y}$.[5] The frontier encodes all information needed for the three checks: `is_cylinder`, `is_member`, and `is_live` can each be evaluated from $\mathcal{F}$ alone, without retaining the full path history (Section 4.3.2, Example 4.3). The pseudocode in Algorithm 4.4 describes how to compute the frontier using the three functions $\mathtt{run}_{\boldsymbol{y}}$, $\mathtt{step}_{\boldsymbol{y}}$, and $\mathtt{closure}_{\boldsymbol{y}}$, and Example 4.3 traces them on a concrete machine.

The design reflects a two-phase structure: before a path's buffer covers the target, the frontier tracks per-state output buffers $(s, \boldsymbol{b})$ filtered by target compatibility. Once the buffer reaches or passes $\boldsymbol{y}$, the frontier can be

[4] In Snæbjarnarson et al. (2026a), this section appears as an appendix.

[5] A buffer $\boldsymbol{b}$ is compatible with $\boldsymbol{y}$ if and only if $\boldsymbol{b} \preceq \boldsymbol{y} \vee \boldsymbol{y} \preceq \boldsymbol{b}$, i.e., the buffer has either covered the target or has yet to diverge from it.

```
def run_y(x_1 ··· x_N):   # memoize
  if N = 0:
    return closure_y(I × {ε})
  return step_y(run_y(x_{<N}), x_N)

def step_y(F, x):
  F' ← ∅
  for (s, b) in F:
    for (_ --x:y--> s') ∈ T(s, x):
      b' ← by
      if b' ⪯ y ∨ y ⪯ b':
        F'.add((s', b'))
  return closure_y(F')

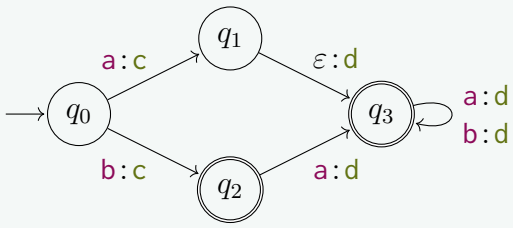
def closure_y(F):
  F' ← F
  P ← QUEUE(F)
  while |P| > 0:
    (s, b) ← P.pop()
    for (_ --ε:y'--> s') ∈ T(s, ε):
      b' ← by'
      if b' ⪯ y ∨ y ⪯ b':
        if (s', b') ∉ F':
          F'.add((s', b'))
          P.add((s', b'))
  return F'
```

Algorithm 4.4: run_y computes the frontier for a source string, step_y advances it by one source symbol, and $\mathsf{closure}_y$ follows ε-input arcs. Each retains only target-compatible buffers.

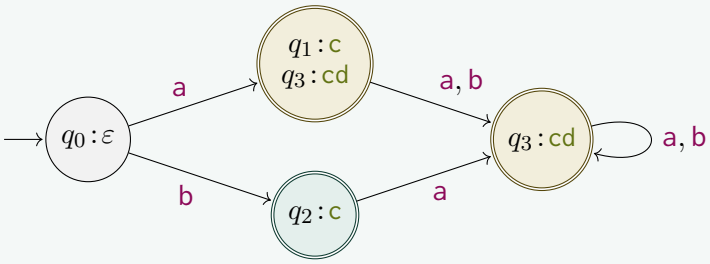
truncated for universality checking (Section 4.3.2). The frontier unifies the two phases into a single lazy data structure.

Example 4.3 — Tracing the frontier computation

Trace the frontier computation on the transducer below, with target $\boldsymbol{y} = \mathtt{c}$. The partial functions in this example and Example 4.4 can be extended to total rational functions without adding outputs that begin with c, so $\mathcal{P}(\mathtt{c})$ is unchanged.



The base case is the ε-closure of the initial states: $\mathtt{run}_{\mathtt{c}}(\varepsilon) = \mathtt{closure}_{\mathtt{c}}(\{(q_0, \varepsilon)\}) = \{(q_0, \varepsilon)\}$, since q_0 has no ε-arcs. Reading a: step follows $q_0 \xrightarrow{\mathtt{a:c}} q_1$, appending the output to the buffer to give $(q_1, \mathtt{c})$. The buffer equals the target, so the compatibility filter keeps it. Then closure fires: $q_1 \xrightarrow{\varepsilon:\mathtt{d}} q_3$ adds $(q_3, \mathtt{cd})$. No input was consumed, but the buffer grew past the target and remains compatible ($\boldsymbol{y} \preceq \mathtt{cd}$). Hence $\mathtt{run}_{\mathtt{c}}(\mathtt{a}) = \{(q_1, \mathtt{c}), (q_3, \mathtt{cd})\}$: the same source string reaches two machine states with different output buffers. Reading b instead reaches only $\mathtt{run}_{\mathtt{c}}(\mathtt{b}) = \{(q_2, \mathtt{c})\}$. For the finite graph below, buffers are truncated to length $|\boldsymbol{y}|+1 = 2$. Its nodes are sets of $(s, \boldsymbol{b})$ pairs and its edges are input symbols.



Advancing $(q_3, \mathtt{cd})$ by any input produces cdd, truncated back to cd, yielding the self-loop. Keeping one symbol beyond the target preserves the committed next output symbol, which decompose_next (Section 4.3.5) reads off the frontier. trunc_buf (Algorithm 4.5) keeps only length $|\boldsymbol{y}|$, which suffices for the cylinder check because symbols past the target position are unconstrained by the compatibility filter. Either bound keeps the state space finite.

is_member holds for both one-symbol source strings, through q_3 for a and q_2 for b. is_cylinder(a, c) runs its BFS over truncated frontiers: on

input a, q_1 has no arc but $(q_3, \texttt{cd})$ steps to a self-loop (inside the BFS, `trunc_buf` carries the length-$|\boldsymbol{y}|$ pair $(q_3, \texttt{c})$, while the graph above also shows the committed symbol d), and likewise on b: every input symbol has a non-empty, accepting successor frontier, so $\texttt{a} \in \mathcal{Q}(\texttt{c})$. For b the BFS fails at once: q_2 has no arc on b, so the successor frontier is empty and $\texttt{b} \in \mathcal{R}(\texttt{c})$. Had a path emitted d before c, the compatibility filter would have discarded it as soon as its buffer diverged from the target. In summary (gold nodes universal, teal accepting but not universal):

$\boldsymbol{x}$	Frontier	`is_live`	`is_cylinder`	`is_member`	Class
ε	$\{(q_0, \varepsilon)\}$	✓	✗	✗	extend
a	$\{(q_1, \texttt{c}), (q_3, \texttt{cd})\}$	✓	✓	—	$\mathcal{Q}(\texttt{c})$
b	$\{(q_2, \texttt{c})\}$	✓	✗ (q_2 dead on b)	✓	$\mathcal{R}(\texttt{c})$

The frontier enjoys a useful monotonicity property: extending the target by one symbol can only remove elements from the frontier, never add new ones. This means we can compute the frontier incrementally, filtering rather than rebuilding, as the target grows.

Proposition 4.1 (Frontier containment)**.** *For any source string $\boldsymbol{x}$, target string $y_1 \cdots y_N$ with $N > 0$:*

$$\texttt{run}_{y_1 \cdots y_N}(\boldsymbol{x}) = \{(s, \boldsymbol{b}) \in \texttt{run}_{y_1 \cdots y_{N-1}}(\boldsymbol{x}) \colon |\boldsymbol{b}| < N \vee \boldsymbol{b}_N = y_N\}$$

Proof. Buffer filtering in `step` and `closure` keeps only $(s, \boldsymbol{b})$ pairs where $\boldsymbol{b}$ and the target agree on their shared prefix. Since FST arcs only append to the buffer, once $\boldsymbol{b}$ is long enough to have an N^{th} symbol and $\boldsymbol{b}_N \neq y_N$, then all descendant buffers would be filtered out by the $y_1 \cdots y_N$-compatibility filter. Therefore, filtering $\texttt{run}_{\boldsymbol{y}}(\boldsymbol{x})$ at the end is equivalent to filtering at each step.

To see this, note that the ($\subseteq$) direction holds because $y_1 \cdots y_N$-compatible elements are also $y_1 \cdots y_{N-1}$-compatible. The ($\supseteq$) direction holds because elements that are $y_1 \cdots y_{N-1}$-compatible but *not* $y_1 \cdots y_N$-compatible (i.e., $|\boldsymbol{b}| \geq N$ and $\boldsymbol{b}_N \neq y_N$) cannot produce descendants that become $y_1 \cdots y_N$-compatible, since buffers only grow. ■

Proposition 4.2 (Generalized frontier containment)**.** *Proposition 4.1 generalizes beyond single-symbol extensions: for any* $\boldsymbol{y}' \preceq \boldsymbol{y}$*:*

$$\mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x}) = \{(s, \boldsymbol{b}) \in \mathsf{run}_{\boldsymbol{y}'}(\boldsymbol{x}) \colon \boldsymbol{b} \preceq \boldsymbol{y} \vee \boldsymbol{b} \succeq \boldsymbol{y}\}$$

Proof. By induction on $|\boldsymbol{y}| - |\boldsymbol{y}'|$. The base case $\boldsymbol{y} = \boldsymbol{y}'$ is immediate. For the inductive step, write $\boldsymbol{y} = \boldsymbol{y}_{<N}\, y_N$ with $\boldsymbol{y}' \preceq \boldsymbol{y}_{<N}$. By Proposition 4.1, $\mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x}) = \{(s, \boldsymbol{b}) \in \mathsf{run}_{\boldsymbol{y}_{<N}}(\boldsymbol{x}) \colon |\boldsymbol{b}| < N \vee \boldsymbol{b}_N = y_N\}$. By the induction hypothesis, $\mathsf{run}_{\boldsymbol{y}_{<N}}(\boldsymbol{x}) = \{(s, \boldsymbol{b}) \in \mathsf{run}_{\boldsymbol{y}'}(\boldsymbol{x}) \colon \boldsymbol{b} \preceq \boldsymbol{y}_{<N} \vee \boldsymbol{b} \succeq \boldsymbol{y}_{<N}\}$. Substituting, $(s, \boldsymbol{b}) \in \mathsf{run}_{\boldsymbol{y}'}(\boldsymbol{x})$ survives both filters iff $\boldsymbol{b}$ agrees with $\boldsymbol{y}$ on all positions up to $\min(|\boldsymbol{b}|, N)$, i.e., $\boldsymbol{b} \preceq \boldsymbol{y} \vee \boldsymbol{b} \succeq \boldsymbol{y}$. ■

Proposition 4.2 is the key property that lets `decompose_next` (Section 4.3.5) warm-start from any target prefix's cached frontiers.

4.3.2 Frontier-based checks

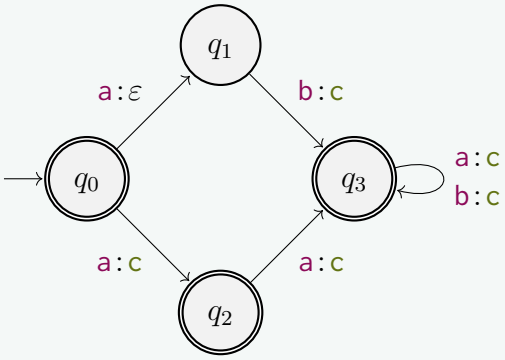
Recall that the decomposition algorithm (Algorithm 4.2) relies on three checks (`is_cylinder`, `is_member`, `is_live`). This section describes how to compute them in terms of the frontier.

To check whether $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$, the frontier-based `is_cylinder` applies `trunc_buf`, which truncates buffers to length $|\boldsymbol{y}|$, to the frontier $\mathsf{run}_{\boldsymbol{y}}(\boldsymbol{x})$, yielding a **truncated frontier**, a set of $(s, \boldsymbol{b})$ pairs whose buffers are compatible with $\boldsymbol{y}$. It then verifies universality via BFS over truncated frontiers: each successor frontier is obtained by applying $\mathsf{step}_{\boldsymbol{y}}$ followed by `trunc_buf`. Unlike buffers in the ordinary frontier, which can grow without bound, truncated buffers must be prefixes of $\boldsymbol{y}$, so at most $|\boldsymbol{y}|+1$ distinct buffers exist, making the truncated frontier state space finite. Each truncated frontier corresponds to a state of the determinized precover machine `P`, so the BFS terminates. The BFS checks that every truncated frontier is both accepting (some pair has $\boldsymbol{b} \succeq \boldsymbol{y}$ and $s \in \mathsf{F}$) and complete (every input symbol leads to a non-empty successor). Without this exploration, two nondeterministic paths may yield the same string yet end in different states s_1 and s_2, where neither is IP-universal on its own, although $[\![\mathsf{f}_{[s_1]}]\!] \cup [\![\mathsf{f}_{[s_2]}]\!] = \mathcal{X}^*$.[6] The precomputation of IP-universal states and a cylinder check shortcut are discussed in Section 4.3.3, and Example 4.4 shows a small transducer where this per-state shortcut fails but the full-frontier BFS succeeds.

[6] Universality is decidable for finite-state automata. E.g., using antichain-based simulation (De Wulf et al., 2006) and bisimulation up to congruence (Bonchi and Pous, 2015).

Example 4.4 — When the per-state shortcut fails

Consider the FST below, where the ε-output arc $q_0 \xrightarrow{\mathtt{a}:\varepsilon} q_1$ creates a frontier element whose output has not yet reached the target.



For target $\boldsymbol{y} = \mathtt{c}$ and source prefix $\boldsymbol{x} = \mathtt{a}$, the frontier is $\mathcal{F} = \{(q_1, \varepsilon), (q_2, \mathtt{c})\}$. The covering states $\{q_2\}$ are not IP-universal (q_2 has no arc on b), so `fast_univ_filter`$(\mathtt{a}, \mathtt{c})$ fails and a would be misclassified as non-quotient. However, q_1 handles input b and produces c, matching the target. The BFS in `is_cylinder` confirms universality over the full frontier, correctly placing a in $\mathcal{Q}(\mathtt{c})$.

When the cylinder check fails, membership reduces to checking whether the frontier contains a covering accepting pair ($\boldsymbol{b} \succeq \boldsymbol{y}$ and $s \in \mathsf{F}$): if so, $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$ and $\boldsymbol{x}$ enters $\mathcal{R}(\boldsymbol{y})$.

If the NFA accepting the precover is trimmed, liveness is exactly non-emptiness of the frontier, even without determinization. Without trimming, non-emptiness gives a conservative liveness check.

4.3.3 IP-universality shortcut

Recall that the cylinder check `is_cylinder` verifies whether every extension of a source prefix maps through the target prefix, using a powerset BFS that can be expensive. A cheap sufficient condition avoids it in many cases: a state $s \in \mathsf{S}$ is **IP-universal** if the input projection of the transducer started from s accepts all of $\mathcal{X}^*$. Intuitively, no matter what input follows, the transducer can still scan it to acceptance. Recall from Section 2.4 the set $\mathsf{U} \subseteq \mathsf{S}$ of IP-universal states.

```
def is_cylinder(x, y):  # memoize
  N ← |y|
  F ← trunc_buf_y(run_y(x))
  V ← {F}; P ← QUEUE({F})
  while P:
    F ← P.pop()
    # Accepting: some pair has matched
    # target and is accepting
    if ¬∃(s, b) ∈ F: b ⪰ y ∧ s ∈ F:
      return False
    # Complete: every input leads to a
    # non-empty successor
    for x' ∈ X:
      F' ← trunc_buf_y(step_y(F, x'))
      if F' = ∅: return False
      if F' ∉ V: V.add(F'); P.push(F')
  return True

def is_member(x, y):
  return ∃(s, b) ∈ run_y(x):
      b ⪰ y ∧ s ∈ F

def is_live(x, y):
  return run_y(x) ≠ ∅

def trunc_buf_y(F):
  N ← |y|
  return {(s, b_≤N) | (s, b) ∈ F, b ⪯ y ∨ y ⪯ b}
```

Algorithm 4.5: Frontier-based checks. `is_cylinder`(x, y) verifies universality via powerset BFS over truncated frontiers. `is_member` checks for a covering accepting pair, and `is_live` checks whether the frontier is non-empty.

Proposition 4.3 (IP-universality shortcut)**.** *If some covering frontier state of* x *with respect to* y *is IP-universal, then* x *is a cylinder:* $\big(\exists\,(s, b) \in \mathsf{run}_y(x) : b \succeq y \wedge s \in \mathsf{U}\big) \Longrightarrow x \in \mathcal{C}(y)$.

Proof. Let $(s, b) \in \mathsf{run}_y(x)$ with $b \succeq y$ and $s \in \mathsf{U}$ be the witness from the antecedent. Then $[\![\mathsf{proj}_{\mathcal{X}}(\mathsf{f}_{[s]})]\!] = \mathcal{X}^*$, so for any $x' \in \mathcal{X}^*$, there exists an

accepting run of $\mathtt{f}$ on $\boldsymbol{x}\boldsymbol{x}'$ passing through s with output prefix $\boldsymbol{y}$. Hence $\boldsymbol{x}\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y})$. Since $\boldsymbol{x}'$ was arbitrary, $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$, i.e., $\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y})$. ■

Algorithm 4.6 shows how to precompute the set of IP-universal states efficiently. We first input-project the transducer and remove ε-transitions (replacing each path $s \xrightarrow{\varepsilon} \cdots \xrightarrow{\varepsilon} s' \xrightarrow{x} s''$ with a direct transition $s \xrightarrow{x} s''$, see, e.g., Hopcroft, Motwani, and Ullman 2001, Ch. 2) to obtain an ε-free NFA $\mathtt{A}$ over $\mathcal{X}$. Both operations preserve the state set. Input projection drops output labels. ε-removal adds shortcuts through ε-closures and marks as accepting any state whose ε-closure contains an accepting state. Hence $\mathsf{S}_{\mathtt{A}} = \mathsf{S}$, and states in U correspond directly to transducer states in the frontier. A state s of $\mathtt{A}$ is IP-universal if it is accepting and, for every $x \in \mathcal{X}$, has at least one x-successor that also passes this sufficient test. We compute the resulting set in linear time using an algorithm analogous to Kahn's algorithm for topological sorting (Kahn, 1962): initialize the candidate set U to the accepting states of $\mathtt{A}$ and maintain per-state, per-symbol successor counts. When a count reaches zero, remove the state and propagate the change to its predecessors via a reverse transition index. Each transition is processed at most once, yielding $\mathcal{O}(|\mathsf{T}_{\mathtt{A}}|)$ time. The shortcut can be integrated into `is_cylinder` (Algorithm 4.5) as a fast path:

```
def is_cylinder(x, y):  # memoize
  if fast_univ_filter(x, y): return True  # fast path
  N ← |y|
  ...  # continue BFS as before
```

For transducers with many IP-universal states, this shortcut can avoid the universality BFS in many cylinder checks. For the token-to-byte transducers used in the experiments, all states are IP-universal, so the universality BFS is never needed.

4.3.4 Pruning

Even when the decomposition is finite, the quotient and remainder can be too large to enumerate, so pruning is essential for scalability. At minimum, `prune` drops source strings with zero prefix probability, since these cannot contribute any mass:

```
def prune(𝒫′):
  return {x ∈ 𝒫′ : p⃗_𝒳(x) > 0}
```

```
# Preprocess: input-project and ε-remove
A ← eps-remove(proj_𝒳(f))
# queue-based greatest fixpoint
U ← F_A;   𝒫 ← ∅
count[·,·] ← 0
for (s -x-> s') ∈ T_A with s' ∈ U:
  count[s, x] += 1
for s ∈ U:
  if ∃x ∈ 𝒳: count[s, x] = 0:
    𝒫.add(s)
while 𝒫 ≠ ∅:
  s' ← 𝒫.pop()
  U ← U \ {s'}
  for (s, x) with (s -x-> s') ∈ T_A:
    count[s, x] -= 1
    if count[s, x] = 0 and s ∈ U:
      𝒫.add(s)

def fast_univ_filter(x, y):
  # Is any covering frontier state IP-universal?
  return ∃(s, b) ∈ run_y(x): b ⪰ y ∧ s ∈ U
```

Algorithm 4.6: Efficient computation of IP-universal states and the `fast_univ_filter` method that uses them.

This simplistic filter drops only strings with *zero* probability. In practice, we use the probability-mass pruning strategy in Algorithm 4.7. Given candidates $\mathscr{P}'$ with total mass $Z = \sum_{\boldsymbol{x} \in \mathscr{P}'} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$, we sort by descending $\overrightarrow{p_{\mathcal{X}}}$ and greedily retain candidates until their cumulative mass reaches $(1 - \tau) \cdot Z$ or their count reaches the capacity $n_{\max}$, whichever comes first. The resulting decomposition is approximate.

Approximation bound. At each step where $n_{\max}$ is not the binding constraint, the discarded mass is at most $\tau \cdot Z$. When $n_{\max}$ cuts the queue first, the discarded mass may exceed this bound, as happens in the DNA experiments of Section 4.4 at tight thresholds. For a finite decomposition, as $\tau \to 0$ and $n_{\max} \to \infty$, the computed prefix probability converges to the exact value.

Bounded queue. Pruning bounds the queue at $n_{\max}$ candidates per step even for transducers whose decomposition is infinite (i.e., outside the sufficient

```
def prune_prob_mass(τ, n_max):
  def prune(𝒫′):
    M ← |𝒫′|
    if M = 0: return ∅
    𝒫′ ← sort_descending(𝒫′, key=p⃗_𝒳)
    w ← [p⃗_𝒳(x) for x ∈ 𝒫′]   # w_1 ≥ ··· ≥ w_M
    W ← cumulative_sum(w)
    Z ← W_M
    𝒫 ← ∅
    for m ∈ {1, … , min(n_max, M)}:
      𝒫.add(𝒫′_m)
      if W_m ≥ (1 − τ)Z: break
    return 𝒫
  return prune
```

Algorithm 4.7: Probability-mass pruning (prune) sorts the queue by prefix probability and retains candidates until their cumulative mass reaches $(1-\tau)\,Z$ or their count reaches $n_{\max}$.

conditions of Section 3.3).

Backtracking. Pruning may occasionally reach a dead end in which the observed next symbol receives zero conditional probability under the approximate $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$. A backtracking procedure attempts to recover by halving the threshold and evicting cached decompositions at exponentially increasing depth (Section A.1.3). Backtracking is not needed in the reported experiments at thresholds $\tau \leq$ 1e-3, and the constants and per-transducer fallback counts are reported in Section A.2.3.

The pruning introduced above can introduce bias. Chapter 5 overcomes this limitation with an unbiased algorithm.

4.3.5 Faster next-symbol decomposition

Since $\langle \mathcal{Q}(\boldsymbol{y}y) \rangle \subseteq \langle \mathcal{Q}(\boldsymbol{y}) \rangle$, we have $\mathcal{P}(\boldsymbol{y}y) = (\langle \mathcal{Q}(\boldsymbol{y}) \rangle \cap \langle \mathcal{Q}(\boldsymbol{y}y) \rangle) \sqcup \mathcal{R}(\boldsymbol{y}y)$. The decomposition for $\boldsymbol{y}y$ can therefore be obtained with incremental work starting from $\mathcal{Q}(\boldsymbol{y}) \cup \mathcal{R}(\boldsymbol{y})$. The structural reason is monotonicity. For each source string, the frontier for $\boldsymbol{y}y$ is obtained by filtering the corresponding frontier for $\boldsymbol{y}$, as formalized by Proposition 4.1.

Without pruning, the `decompose_next` method computes $\mathcal{Q}(\boldsymbol{y}y)$ and $\mathcal{R}(\boldsymbol{y}y)$ for all $y \in \mathcal{Y}$ simultaneously, as well as the exact preimage $f^{-1}(\boldsymbol{y})$. This is the key efficiency improvement over calling `decompose`$(\boldsymbol{y}y)$ independently for each y.

The BFS is seeded with $\mathcal{Q}(\boldsymbol{y}) \cup \mathcal{R}(\boldsymbol{y})$ (Section 4.3.5). For each source string $\boldsymbol{x}$ in the queue, we first check whether `is_exact_member`$(\boldsymbol{x}, \boldsymbol{y})$ (Section 4.3.5) and, if so, record $\boldsymbol{x}$ in the preimage set. Next, for each $y \in$ `reachable_outputs`$(\boldsymbol{x}, \boldsymbol{y})$ (defined in Algorithm 4.8), we classify $\boldsymbol{x}$ as a cylinder or member for $\boldsymbol{y}y$. By functionality of `f`, at most one y can be a cylinder (Proposition 4.5). If a cylinder was found for some y, the entire subtree rooted at $\boldsymbol{x}$ is absorbed: we skip extension for *all* symbols (Section 4.3.5). Otherwise, we extend by all reachable source symbols (Section 4.3.5).

We next describe target recursion for cheap frontier computation and three frontier-based helpers.

Target recursion. The call `is_cylinder`$(\boldsymbol{x}, \boldsymbol{y}y)$ triggers `run`$_{\boldsymbol{y}y}(\boldsymbol{x})$. Target recursion is used opportunistically when the parent frontier `run`$_{\boldsymbol{y}}(\boldsymbol{x})$ is already cached: the frontier for the extended target is derived by filtering rather than recomputing the entire source-string chain (Proposition 4.1). This is the typical case in `decompose_next`, where `decompose`$(\boldsymbol{y})$ has already populated the cache. This optimization is transparent: `is_cylinder` is called unchanged. The logic lives entirely inside `run`.

Per-symbol frontier helpers. The BFS uses three frontier-based helpers.

- `is_exact_member`$(\boldsymbol{x}, \boldsymbol{y})$ checks whether $\boldsymbol{x}$ maps to $\boldsymbol{y}$ exactly, $f(\boldsymbol{x}) = \boldsymbol{y}$.
- `reachable_inputs`$(\boldsymbol{x}, \boldsymbol{y})$ conservatively approximates the set of source symbols x' such that extending $\boldsymbol{x}$ by x' remains compatible with $\boldsymbol{y}$, i.e., $\{x' \in \mathcal{X} \colon \langle \boldsymbol{x}x' \rangle \cap \mathcal{P}(\boldsymbol{y}) \neq \emptyset\}$. The pseudocode checks for transitions from frontier states whose extended buffer is prefix-compatible with $\boldsymbol{y}$. This may over-report when the resulting state has no target-compatible accepting continuation.
- `reachable_outputs`$(\boldsymbol{x}, \boldsymbol{y})$ conservatively approximates the set of target symbols y' such that some extension of $\boldsymbol{x}$ can produce $\boldsymbol{y}y'$ as an output prefix, i.e., $\{y' \in \mathcal{Y} \colon \langle \boldsymbol{x} \rangle \cap \mathcal{P}(\boldsymbol{y}y') \neq \emptyset\}$. The pseudocode checks single-step transitions from frontier states, which may over-report when the resulting state has no target-compatible accepting continuation. Symbols requiring multiple transitions to extend past $\boldsymbol{y}$ are handled by the expansion loop.

All three are computed directly from the frontier `run`$_{\boldsymbol{y}}(\boldsymbol{x})$:

```
def is_exact_member(x, y):
  return ∃(s, b) ∈ run_y(x):
    b = y ∧ s ∈ F

def reachable_inputs(x, y):
  return {x′ | (s, b) ∈ run_y(x),
    (s --x′:y′--> s′) ∈ T, by′ ⪯ y ∨ y ⪯ by′}

def reachable_outputs(x, y):
  F ← run_y(x)
  result ← ∅
  for (s, b) ∈ F:
    if |b| > |y|:
      result.add(b_{|y|+1})
    for (s --:y--> s′) ∈ T(s):
      for y′ ∈ Y:
        if by ⪯ yy′ or yy′ ⪯ by:
          result.add(y′)
  return result
```

Algorithm 4.8: Frontier helpers used by `decompose_next` (Algorithm 4.9): exact membership and the conservative input/output reachability filters.

Expansion loop. The following algorithm combines the seeding, shortcuts, and helpers above into a single BFS that computes the per-symbol quotients and remainders for every $y \in \mathcal{Y}$ simultaneously.

`decompose_next`$(\boldsymbol{y})$ computes $\mathcal{Q}(\boldsymbol{y}y)$ and $\mathcal{R}(\boldsymbol{y}y)$ for all $y \in \mathcal{Y}$ in a single BFS pass. The `is_cylinder`$(\boldsymbol{x}, \boldsymbol{y}y)$ call triggers `run`$_{\boldsymbol{y}y}(\boldsymbol{x})$, which is computed cheaply via target recursion (Proposition 4.1) since the parent frontier `run`$_{\boldsymbol{y}}(\boldsymbol{x})$ is already cached. The non-cylinder shortcut avoids the universality check entirely for seeds in $\mathcal{R}(\boldsymbol{y})$. After the BFS, the cache is populated so that subsequent `decompose`$(\boldsymbol{y}y)$ calls are free. The exact preimage is collected as a byproduct: if $\boldsymbol{x}$ is a cylinder for some $\boldsymbol{y}y$, every extension produces output strictly longer than $\boldsymbol{y}$, so no extension can be an exact preimage of $\boldsymbol{y}$.

Cache population. After the BFS, the computed $\mathcal{Q}(\boldsymbol{y}y)$ and $\mathcal{R}(\boldsymbol{y}y)$ should be written into `decompose`'s memoization cache, so that a subsequent call to `decompose`$(\boldsymbol{y}y)$ (e.g., from `decompose_next`$(\boldsymbol{y}y)$) hits the cache instead of re-running its own BFS.

```
def decompose_next(y):
  (Q, R) ← decompose(y)
  qs ← {}; rs ← {}          # map y ↦ set of x
  ps ← ∅
  # Remainders were not cylinders for y,
  # hence not for any yy (Proposition 4.4)
  non_cyl ← R
  P ← Q ∪ R
  while |P| > 0:
    P' ← ∅
    for x in P:
      if is_exact_member(x, y) : ps.add(x)
      # f(x) = y exactly
      # Check reachable y; at most one y
      # can have x as a cylinder (Proposition 4.5).
      ŷ ← ⊥
      for y in reachable_outputs(x, y) :
        if ŷ = ⊥ and x ∉ non_cyl and is_cylinder(x, yy):
          qs[y].add(x); ŷ ← y; continue
        if is_member(x, yy):
          rs[y].add(x)
      if ŷ ≠ ⊥:
        # absorbed ⇒ skip extension for all y
        continue
      for x' ∈ reachable_inputs(x, y) :
        P'.add(xx')
    P ← prune(P')
  return (qs, rs, ps)
```

Algorithm 4.9: `decompose_next` jointly computes the quotient and remainder for every one-symbol extension of a target prefix in a single BFS pass.

```
def decompose_next(y):
  ...
  # Populate decompose's cache for all children
  for y ∈ qs.keys() ∪ rs.keys():
    cache[decompose, yy] ← (qs[y], rs[y])
  return ...
```

4.3.6 Next-symbol decomposition shortcuts

Starting with the decomposition from the prior step and using target recursion lets `decompose_next` reuse frontier computations. The remaining bottleneck is the per-symbol cylinder check `is_cylinder` (Algorithm 4.5). Three shortcuts avoid or reduce this cost, each justified by a proposition about the structure of the precovers. *Non-cylinder monotonicity* and *cylinder uniqueness* enter Algorithm 4.9 directly (Section 4.3.5). The *combined-universality* shortcut can sometimes establish that a source prefix is a quotient element for an extended target without running the full cylinder check. It is described in Section A.1.1.

Non-cylinder shortcut. Strings in $\mathcal{R}(\boldsymbol{y})$ cannot be quotient elements for $\boldsymbol{y}$. And since $\langle \mathcal{Q}(\boldsymbol{y}y') \rangle \subseteq \langle \mathcal{Q}(\boldsymbol{y}) \rangle$, they are not quotient elements for the strictly more restrictive $\boldsymbol{y}y$ either. The expensive cylinder check (Section 4.3.5) is thus skipped for these seeds (Section 4.3.5).

Proposition 4.4 (Non-cylinder monotonicity)**.** *If* $\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})$ *then* $\boldsymbol{x} \notin \mathcal{Q}(\boldsymbol{y}y)$ *for all* $y \in \mathcal{Y}$.

Proof. We have that $\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})$ implies $\langle \boldsymbol{x} \rangle \not\subseteq \mathcal{P}(\boldsymbol{y})$. Since $\mathcal{P}(\boldsymbol{y}y) \subseteq \mathcal{P}(\boldsymbol{y})$, we have $\langle \boldsymbol{x} \rangle \not\subseteq \mathcal{P}(\boldsymbol{y}y)$, so $\boldsymbol{x} \notin \mathcal{Q}(\boldsymbol{y}y)$. ■

Cylinder uniqueness shortcut. Since the transformations we consider are functions, at most one output symbol y can have $\boldsymbol{x}$ as a cylinder element. Once a cylinder element is found, the cylinder check is skipped for the remaining output symbols (Section 4.3.5).

Proposition 4.5 (Cylinder uniqueness)**.** *For a function* f *and target prefix* $\boldsymbol{y}$, $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}y) \cap \mathcal{Q}(\boldsymbol{y}y') \Longrightarrow y = y'$.

Proof. Suppose $y \neq y'$ and $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}y) \cap \mathcal{Q}(\boldsymbol{y}y')$. Then $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y}y) \cap \mathcal{P}(\boldsymbol{y}y') = f^{-1}(\boldsymbol{y}y\mathcal{Y}^*) \cap f^{-1}(\boldsymbol{y}y'\mathcal{Y}^*) = \emptyset$, where the last equality holds because f is a function and $\boldsymbol{y}y\mathcal{Y}^* \cap \boldsymbol{y}y'\mathcal{Y}^* = \emptyset$. This contradicts $\langle \boldsymbol{x} \rangle \neq \emptyset$. ■

4.3.7 Memoization and batching of $p_{\mathcal{X}}$

The source LM is queried in two places: first during decomposition, where probability-mass pruning calls $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ to rank candidates, and then during

scoring, where `prefix_prob` and `next_dist` sum $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ and $p_{\mathcal{X}}(\boldsymbol{x})$ over the quotient and remainder. These calls involve many overlapping source prefixes. Naïvely, each call recomputes the entire chain $\overrightarrow{p_{\mathcal{X}}}(x_1) \cdot \overrightarrow{p_{\mathcal{X}}}(x_2 \mid x_1) \cdots$ from scratch. Memoizing the conditional $\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$ avoids this: given a cached state for $\boldsymbol{x}$, extending by one symbol x requires a single conditional evaluation rather than replaying the entire history.

For transformer language models, the cached state is the *key–value (KV) cache*: the accumulated key and value tensors from the self-attention layers. Extending by one token reuses the cached KV entries and runs a single forward pass for the new position. Most LMs return the complete next-symbol distribution $\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$ in a single call, so we cache the full distribution rather than individual values. When the extension loop generates children $\boldsymbol{x}x_1, \ldots, \boldsymbol{x}x_k$, their prefix probabilities are read directly from the cached distribution without additional LM calls.[7]

Batching while pruning. The `prune` step (Section 4.3.4) needs $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ for every string $\boldsymbol{x}$ in the queue. Many of these strings share a common prefix but extend by different symbols. We group the queue by parent: for each parent $\boldsymbol{x}_{<N}$ whose next-symbol distribution $\overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x}_{<N})$ is not yet cached, we issue a single batched LM call. Each call returns the full distribution, from which $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}_{<N}x) = \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}_{<N}) \cdot \overrightarrow{p_{\mathcal{X}}}(x \mid \boldsymbol{x}_{<N})$ is read off for all children simultaneously. On GPU-based LMs, batching these parent contexts into a single forward pass amortizes the per-call overhead.

4.3.8 Realizing the autoregressive interface

Given the decomposition, the transduced LM interface from Section 4.1 is implemented as follows. The prefix probability $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$ sums LM prefix probabilities over `Q` and LM string probabilities over `R`. The string probability $p_{\mathcal{Y}}(\boldsymbol{y})$ sums over the exact preimage. The next-symbol distribution $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$ is computed via `decompose_next`, which provides the per-symbol decompositions and exact preimage in a single BFS pass. The computational cost is dominated by `decompose_next`, and once the per-symbol quotients, remainders, and preimage are available, the scoring reduces to lookups into the memoized source LM (Section 4.3.7).

The naïve next-symbol distribution computes `prefix_prob`$(\boldsymbol{y}y)$ separately for each y (see Section 4.1). The reference implementation is correct, but inefficient because it does not share the BFS across output symbols. Each

[7] Our implementation uses `genlm` (`https://github.com/genlm/genlm-backend`) for its caching and auto-batching interface.

```
def prefix_prob(y):
  (Q, R) ← decompose(y)
  return Σ_{x∈Q} p⃗_X(x) + Σ_{x∈R} p_X(x)

def prob(y):
  (_, _, ps) ← decompose_next(y)
  return Σ_{x∈ps} p_X(x)

def next_dist(y):
  (qs, rs, ps) ← decompose_next(y)
  p̄ ← {}
  for x ∈ ps: p̄[EOS] += p_X(x)
  for y ∈ qs.keys() ∪ rs.keys():
    for x ∈ qs[y]: p̄[y] += p⃗_X(x)
    for x ∈ rs[y]: p̄[y] += p_X(x)
  Z ← Σ_{y∈Y⊔{EOS}} p̄[y]
  if Z = 0: return {EOS : 1}
  return p̄/Z
```

Algorithm 4.10: `next_dist` uses `decompose_next` to compute and normalize all next-symbol probabilities in one BFS pass.

`prefix_prob`(yy') call triggers its own `decompose` pass and cannot use the cross-symbol shortcuts described in Section 4.3.5.

The following fast path specializes `next_dist` for the transducer class used in the token-to-byte experiments.

4.3.9 All-universal fast path

When every transducer state is IP-universal, the remainder is always empty. If, additionally, every non-ε-input arc produces at least one output symbol, then every newly discovered live extension is immediately classified as a quotient element, so decomposition requires at most one source-symbol expansion. Boundary frontier elements can then be resolved in closed form via a precomputed first-output table. Together, these conditions yield the all-universal fast path, a specialization of `next_dist` (Algorithm 4.10). The specialization is described in Section A.1.2. The $\mathtt{f}_\alpha$ transducers satisfy both preconditions (Table A.4: all states IP-universal, no ε-output on non-ε-input arcs), which helps explain their substantially higher throughput (Section 4.4). The DNA transducer $\mathtt{f}_{\text{dna2aa}}$ has all states IP-universal but uses ε-output arcs (the first

two bases of each codon emit no amino acid), so the fast path does not apply.

4.3.10 Label pushing

As a one-time preprocessing step (see, e.g., Mohri, 1997; Oncina, Garcia, and Vidal, 1993), we push output labels toward the initial state of $\mathtt{f}$: when every path through a state s emits output beginning with y, that y is shifted to the incoming arc so it is produced one step earlier. This ensures that output is committed as early as possible, speeding up the frontier-based cylinder check.

4.4 Experiments

We evaluate the algorithms on three transducer families spanning the prefix-monotonicity cases of proposition 3.1: token-to-byte (strict-prefix monotone), DNA-to-amino-acid (prefix monotone with multi-symbol lookahead), and PTB tokenization (not prefix monotone). For each use case, we follow the approach in Vieira et al. (2025a) and measure the Jensen–Shannon divergence (JSD) between distributions obtained with probability-mass pruning (Section 4.3.4) and a reference obtained with the tightest pruning threshold. This reference is itself an approximation rather than the exact distribution, so a small JSD verifies agreement with it but does not provide a bound on the divergence from the exact transduced LM. Chapter 5 provides unbiased estimates for assessing pruning error. We also report cross-entropy loss (Section A.3.4), which reflects the cost of scoring specific sequences rather than full distributions. Experiments use GPT-2 Large (p_{gpt2}) (Radford et al., 2019), Llama 3.2-1B (p_{llama1B}) and Llama 3.1-8B (p_{llama8B}) (Grattafiori et al., 2024), Phi-4 (p_{phi4}) (14B; Abdin et al., 2024), and a DNA model trained on the human genome (p_{dna}). For the experiments in Section 4.4, we use `genlm-bytes`[8] to convert token-level models to byte-level models. See Section A.2 for details on the training and evaluation setup, and Section A.2.5 for transducer details.

Recall that a transduced language model $p_{\mathcal{Y}}$ is characterized by a source model $p_{\mathcal{X}}$ and a transducer $\mathtt{f}$ encoding some function f (Section 3.1). To make this dependency explicit, we write $p_{\mathcal{X}} \circ \mathtt{f} \stackrel{\text{def}}{=} p_{\mathcal{Y}}$ and refer to the operation as composing. All experiments compute next-symbol distributions using the more efficient algorithm (Section 4.3.8), which reuses cached decompositions across target positions and exploits structural shortcuts (Section 4.3.6). Probability-mass pruning (Section 4.3.4) retains the most likely candidates

[8] `genlm-bytes` is an implementation of Vieira et al. (2025a). See `https://github.com/genlm/genlm-bytes`.

until the retained mass reaches its prescribed level. The DNA experiments additionally impose a candidate-set cap.

From tokens to bytes. We revisit the algorithm for converting models from tokens to bytes, as in Vieira et al. (2025a). This transformation can be realized by a simple transducer $\mathtt{f}_\alpha$, with $\mathcal{O}(|\mathcal{X}|)$ states. Specifically, the transducer contains a chain for each token in the input vocabulary $\mathcal{X}$. This structure allows us to use the all-universal fast path (Section 4.3.9), which resolves each quotient element with at most one LM call. An example of such a machine is given in Figure 4.1. We evaluate the efficient algorithm of Section 4.3.8 using the transducers $p_{\mathsf{gpt2}} \circ \mathtt{f}_\alpha$, $p_{\mathsf{llama1B}} \circ \mathtt{f}_\alpha$, $p_{\mathsf{llama8B}} \circ \mathtt{f}_\alpha$, and $p_{\mathsf{phi4}} \circ \mathtt{f}_\alpha$ on the first ten paragraphs of the `wikitext-2-raw-v1` dataset (Merity et al., 2017) (corresponding to the first 7684 bytes). As shown in Figure 4.3 (left) and Table A.8 in Section A.3, lower pruning thresholds (τ) give lower JSD values against a reference distribution ($\tau =$ 1e-5) at the cost of throughput, measured in bytes per second. We confirm that the JSD values are similar to those achieved by Vieira et al. (2025a) in Section A.3.1, Table A.6.

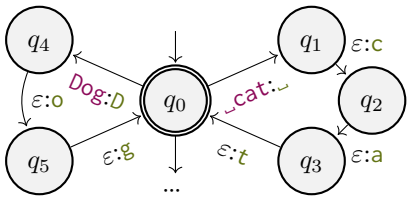


Figure 4.1: Part of a token-to-character FST, paths shown for `␣cat` and `Dog`.

From tokens to orthographic word boundaries. The next transformation we consider is converting language models over tokens to language models over *orthographic* words. The precise definition of such a word varies with the application. In some settings, contractions such as "`wouldn't`" should be treated as a single unit. In other settings, however, a more natural segmentation would be "`would`" and "`n't`". We can accommodate any FST-based definition. Linguistic tokenizers, such as the PTB tokenizer (Marcus, Santorini, and Marcinkiewicz, 1993), use contextual information to segment raw English text into linguistically meaningful units suitable for downstream NLP tasks. We construct an FST that encodes the PTB tokenizer, $\mathtt{f}_{\mathrm{ptb}}$, with details in Section A.2.5. One rule of the transducer, shown in Figure 4.2, inserts SEP before a comma if a digit does not follow. Composing $p_{\mathcal{X}} \circ \mathtt{f}_\alpha \circ \mathtt{f}_{\mathrm{ptb}}$ yields a transduced language model over characters and separators marking PTB token boundaries. To reduce the state count of the composed FST and improve efficiency,

we represent $p_{\mathcal{X}} \circ \mathtt{f}_{\alpha}$ using the byte-transformed models from `genlm-bytes`. In Figure 4.3 (right), we plot the average JSD against the throughput for different thresholds τ, using the same dataset as in Section 4.4. The reference distribution uses $\tau =$ 1e-4 (3e-4 for p_{phi4}). As with the experiments in Section 4.4, we observe lower JSD at lower thresholds, albeit at the cost of throughput. For experiments using higher thresholds, see Section A.3, Table A.9.

Converting DNA models to models over amino acids. We use a transducer that converts sequences of the four DNA nucleobases to sequences over the 23-symbol target alphabet.[9] Let $p_{\mathsf{dna}} \circ \mathtt{f}_{\mathrm{dna2aa}}$ denote the transduced model that converts DNA nucleobases into amino acids. To evaluate our approach, we sample 65 proteins.[10] In Tables A.10 and A.11, we show the average JSD and throughput for different thresholds τ. This setting is particularly challenging because DNA language models typically assign non-negligible mass to many candidates in the decomposed precover. The number of candidates can grow exponentially with the sequence length. To mitigate the combinatorial blow-up, we cap the candidate set and report throughput and JSD while varying the cap.

The full experimental setup and additional quantitative results are collected in Chapter A (Sections A.2 and A.3).

[9] I.e., $\mathcal{X} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$ and $\mathcal{Y} = \{\mathtt{A}, \mathtt{R}, \mathtt{N}, \mathtt{D}, \mathtt{C}, \mathtt{Q}, \mathtt{E}, \mathtt{G}, \mathtt{H}, \mathtt{I}, \mathtt{L}, \mathtt{K}, \mathtt{M}, \mathtt{F}, \mathtt{P}, \mathtt{S}, \mathtt{T}, \mathtt{V}, \mathtt{W}, \mathtt{Y}, \mathtt{B}, \mathtt{Z}, *\}$.

[10] Retrieved from `https://www.uniprot.org/uniprotkb`; see Table A.1 for the accession numbers.

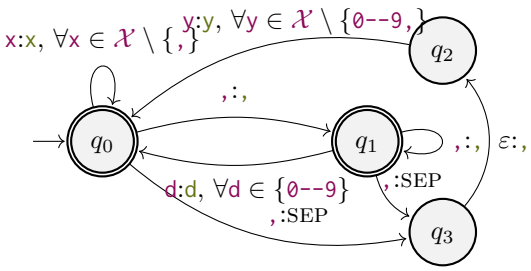


Figure 4.2: An FST that inserts a separator (SEP) before commas followed by non-digit characters.

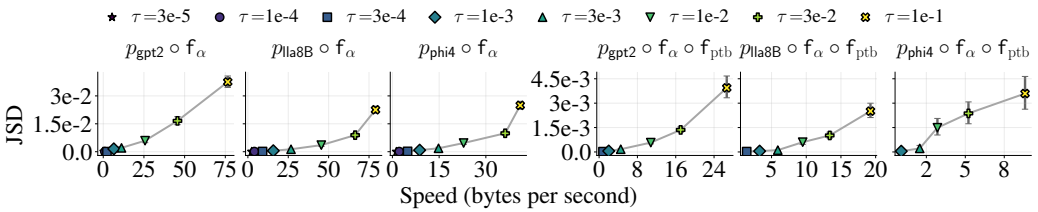


Figure 4.3: Average Jensen–Shannon divergence (JSD) and throughput (bytes/sec) across thresholds τ. Left: $p_{\mathcal{X}} \circ \mathtt{f}_\alpha$ (reference: τ = 1e-5). Right: $p_{\mathcal{X}} \circ \mathtt{f}_\alpha \circ \mathtt{f}_{\text{ptb}}$ (reference: τ = 1e-4, and 3e-4 for p_{phi4}). Error bars show 95% bootstrapped CIs. At the tightest thresholds, large decomposition sizes prevent some models from completing all paragraphs (see Section A.3.2).

Chapter 5

Unbiased inference via stochastic estimation

This chapter contains material reproduced and adapted from Snæbjarnarson et al. (2026b), with editorial changes for narrative consistency.

The decomposition-based inference of Chapter 4 is exact when the quotient–remainder decomposition can be fully enumerated, and falls back on beam summing with threshold pruning when that decomposition is too large or infinite. Pruning is convenient but underestimates the target probability whenever it discards mass.

The resulting error is deterministic, so repetition reproduces the same underestimate rather than averaging the error away. Each next-symbol probability is a ratio of two such underestimates, so its error is not even reliably signed. Measuring the exact bias generally amounts to solving the original problem, and the threshold-pruned estimate does not quantify its pruning error. Finally, beam summing with threshold pruning becomes prohibitively expensive on the high-entropy DNA sources studied here (Section 5.2.3). In the uniform-source worst case of Section B.4, a fixed top-K beam captures an exponentially vanishing share of the target mass, so probability-mass pruning must retain exponentially many source strings to preserve any fixed share.

This chapter resolves those issues with an unbiased stochastic alternative to threshold pruning. The target prefix probability is the source mass of the strings whose transduction covers the target.[1] We estimate it by sequential Monte Carlo (SMC) (Doucet, Freitas, and Gordon, 2001; Naesseth, Lindsten, and Schön, 2019). At each target position, the algorithm enumerates source-

[1]When this probability is positive, conditioning the source model on $\boldsymbol{y} \preceq f(X)$ gives $\Pr_{X \sim p_{\mathcal{X}}}[X = \boldsymbol{x} \mid \boldsymbol{y} \preceq f(X)] = p_{\mathcal{X}}(\boldsymbol{x}) \mathbb{1}\{\boldsymbol{y} \preceq f(\boldsymbol{x})\} / \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$. The denominator is the sum of the numerator over $\boldsymbol{x}$, so in standard SMC terminology $\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$ is a normalizing constant.

prefix extensions that can still cover the target, then prunes them stochastically and reweights the survivors. The resulting estimator is unbiased provided the run halts almost surely, so its error can be measured as variance, which decreases as the particle count grows in our experiments.

A standard SMC construction collects mass only when a particle happens to draw EOS at a member prefix. In the reported results on encyclopedic text with the PTB transducer, estimated log probabilities vary at least an order of magnitude more across seeds for the with-replacement SMC baselines than for adaptive-budget SWOR (Table 5.2). On DNA-to-amino-acid transduction, adaptive-budget SWOR reaches 200 amino acids in about five seconds, while beam summing becomes prohibitively slow after a dozen (Section 5.2.3).

5.1 Unbiased estimation of transduced language models

The quotient–remainder decomposition (Equation (3.6)) writes the prefix probability $\overrightarrow{p_Y}(\boldsymbol{y})$ of a transduced language model (Chapter 3) as a sum of source-model mass over the precover $\mathcal{P}(\boldsymbol{y})$. That sum is often far too large to enumerate, so the beam summing of Chapter 4 applies $\texttt{prune}_\tau$, discarding mass to keep the calculation manageable. We therefore replace deterministic pruning with resampling without replacement among the source prefixes produced at each extension step, using *inclusion-probability* reweighting to preserve their weights in expectation (Section 5.1.2). We write $Z \stackrel{\text{def}}{=} \overrightarrow{p_Y}(\boldsymbol{y})$ for this estimand and seek an estimate $\hat{Z}$ that meets three criteria:

(i) it is unbiased, i.e., $\mathbb{E}[\hat{Z}] = Z$
(ii) it has low variance
(iii) it comes at a reasonable and configurable computational cost

As a consequence of (iii), we require that the algorithms halt *almost surely* (with probability 1). This lets us estimate prefix probabilities whose quotient–remainder decomposition (Chapter 3) is infinite, something the enumeration of Chapter 4 cannot guarantee.

We build this estimator starting with the algorithm of Chapter 4, which computes the decomposition of Chapter 3, where $\texttt{prune}_\tau$ is its only lossy step (Section 5.1.1). We then replace $\texttt{prune}_\tau$ with **sampling without replacement** (**SWOR**) (Section 5.1.2), which samples which distinct source prefixes survive rather than discarding them deterministically. The algorithm adds a prefix's quotient contribution when the `is_cylinder` predicate holds and, otherwise, its remainder contribution when the `is_member` predicate holds. Adding the remainder contribution directly rather than sampling the stop

decision is a variance-reducing Rao–Blackwellization. We extend the SWOR-based pruning function to use the running total to reduce the number of retained source prefixes as mass is accumulated, saving compute and ensuring almost-sure halting (Section 5.1.4).

5.1.1 Beam summing over the precover decomposition

The algorithm in Algorithm 5.1 is the beam summing approach of Chapter 4 made to return the estimand directly, here named `beam_summing`. The recursion computes the quotient–remainder decomposition (Equation (3.6)), one call per target prefix. It adds cylinder elements to `Q` (**quotient contributions**) and member stop branches to `R` (**remainder contributions**). We call the queue's entries **particles**, each a source prefix $\boldsymbol{x}$ carrying a nonnegative **weight** w, and the queue itself the **pool**. The queues and the collections `Q` and `R` are multisets. With no pruning, a particle's weight is exactly its source prefix mass $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ (Section 5.1.1). Within each call, `beam_summing` enumerates source prefixes breadth-first. The practical implementation in Chapter 4 uses $\texttt{prune}_\tau$ to keep the enumeration manageable by deterministically keeping at least $1-\tau$ of the mass, for some threshold $0 < \tau < 1$, potentially biasing the estimate by discarding positive covering mass (Table 5.1, Algorithm 5.1). We leave out its additional candidate cap $n_{\max}$.

Each nonempty call to `beam_summing` first recurses on the target prefix one position shorter, then initializes its pool from the returned quotient entries and passes the returned remainder entries through a membership re-check (Section 5.1.1). Within a call, each iteration processes every particle in the pool, forms child particles by extending non-cylinder prefixes by one source symbol, and updates the running total A using the predicate checks (Section 5.1.1). It is passed to `prune` after each iteration, and when the pool empties, the call appends A to the returned list $\hat{Z}$ (Equation (5.2)). The algorithm in Chapter 4 enqueues only source prefixes that pass the Live predicate. `beam_summing` generalizes this restriction through a **twist** ψ, a nonnegative weighting on source prefixes.[2] At every position t, a twist must be positive wherever the optimal twist ψ_t^* defined below is positive, so it zeroes only children with no covering mass ahead. The extension step only creates child particles with $\psi(\boldsymbol{x}x') > 0$ (Section 5.1.1). The stochastic prunes introduced below use the twisted weights $w\,\psi(\boldsymbol{x})$ to choose survivors (Section 5.1.2). Setting $\psi = \texttt{is_live}$ and using $\texttt{prune}_\tau$ as the pruning function recovers the

[2]The name follows the twisted particle filters of Whiteley and Lee (2014), a look-ahead factor multiplying the weights a filter tracks (Heng et al., 2020; Zhao et al., 2024).

```
def beam_summing(y, ψ, prune):
  N ← |y|
  if N = 0:
    (Q′, R′), Ẑ ← ({(ε, 1)}, ∅), []
  else:
    (Q′, R′), Ẑ ←
      beam_summing(y_{<N}, ψ, prune)
  P ← QUEUE(Q′)
  (Q, R), A ← (∅, ∅), 0
  for (x, w) ∈ R′:
    if is_member(x, y):
      R.add((x, w))
      A += w
  while |P| > 0:
    P′ ← ∅
    for x, w ∈ P:
      if is_cylinder(x, y):
        Q.add((x, w))
        A += w
        continue
      if is_member(x, y):
        R.add((x, w · p→_X(EOS | x)))
        A += w · p→_X(EOS | x)
      for x′ ∈ X with
          ψ(xx′) > 0:
        P′.add((xx′, w · p→_X(x′ | x)))
    P ← prune(P′, A)
  Ẑ.append(A)
  return (Q, R), Ẑ

def prune_M(P′):
  # Standard beam search
  ((x^(i), w^(i)))_{i=1}^N ←
    sort_descending(P′, key=w)
  return {(x^(i), w^(i))
    for i = 1, ..., min(M, N)}

def prune_τ(P′):
  # Keep heaviest particles holding
  # ≥ 1 − τ of the mass
  if |P′| = 0: return ∅
  ((x^(i), w^(i)))_{i=1}^N ←
    sort_descending(P′, key=w)
  W_i ← Σ_{j≤i} w^(j)
    for i = 1, ..., N
  k ← smallest i with
    W_i ≥ (1 − τ) W_N
  return {(x^(i), w^(i))
    for i = 1, ..., k}
```

Algorithm 5.1: beam_summing estimates target-prefix probabilities by breadth-first enumeration. The twist restricts extensions, and each prune call receives the pool and the current-position running total A. $\texttt{prune}_M$ retains the M heaviest particles, while $\texttt{prune}_\tau$ retains the heaviest particles holding at least $1-\tau$ of the pool's mass.

uncapped form of the algorithm in Chapter 4.

At position t, let $\texttt{Q}_t$ and $\texttt{R}_t$ be the quotient and remainder entries collected by the call for $\boldsymbol{y}_{\leq t}$. Grouping these entries by source prefix and dividing the remainder entries by their stop factor defines

$$V_t(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{(\boldsymbol{x}', w) \in \texttt{Q}_t} w \, \mathbb{1}\{\boldsymbol{x}' = \boldsymbol{x}\} \;+\; \sum_{(\boldsymbol{x}', w) \in \texttt{R}_t} \frac{w}{\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})} \mathbb{1}\{\boldsymbol{x}' = \boldsymbol{x}\}, \tag{5.1}$$

the per-prefix estimate of $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ produced by the call for $\boldsymbol{y}_{\leq t}$. A remainder summand is defined to be zero when $\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) = 0$, in which case its contribution to A is also zero. Since the quotient and remainder entry weights

are the only terms added to A, grouping them by source prefix gives

$$\hat{Z}_t \stackrel{\text{def}}{=} \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y}_{\leq t})} V_t(\boldsymbol{x})\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \quad + \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}_{\leq t})} V_t(\boldsymbol{x}), \tag{5.2}$$

the quotient–remainder decomposition Equation (3.6) with each exact prefix mass replaced by V_t (Equation (2.1)).

Table 5.1: The TLM algorithms compared in this chapter. `beam_summing` uses all-child extension, and its prune-free form is exact whenever it halts. `smc_simple` and `smc_rb` sample one child per particle; `smc_simple` also samples the stop decision, while `smc_rb` adds quotient and remainder contributions directly. `beam_swor` and `beam_swor_adaptive` combine all-child extension with sampling without replacement. The qualitative variance labels summarize Table 5.2.

Algorithm	Prune	(i) Unbiased	(ii) Variance (qual.)	(iii) Particle count
`beam_summing` (Algorithm 5.1)	none	✓	0	unbounded
↪ + $\texttt{prune}_M$	keep M heaviest	✗	0	fixed M
↪ + $\texttt{prune}_\tau$	keep $\geq 1-\tau$ mass	✗	0	mass-set
`smc_simple` (Section B.1)	multinomial	✓	high	fixed M
`smc_rb` (Section B.1)	multinomial	✓	lower	fixed M
`beam_swor`	SWOR	✓	lowest	fixed M
`beam_swor_adaptive`	SWOR, adaptive	✓	lowest	adaptive, $m \leq M$

What beam summing computes. Fix the target $\boldsymbol{y}$ and define the **optimal twist** of a source prefix as the probability that a completion (Section 2.2) covers the target,

$$\psi^*(\boldsymbol{x}) \stackrel{\text{def}}{=} \Pr_{X \sim p_{\mathcal{X}}}[\boldsymbol{y} \preceq f(X) \mid \boldsymbol{x} \preceq X] = \mathop{\mathbb{E}}_{\boldsymbol{x}'' \sim p_{\mathcal{X}}(\cdot \mid \boldsymbol{x})}[\mathbb{1}\{\texttt{is_member}(\boldsymbol{x}\,\boldsymbol{x}'', \boldsymbol{y})\}]. \tag{5.3}$$

It is optimal because weighting the source conditionals by it yields the source model conditioned on covering $\boldsymbol{y}$. Under the corresponding importance weighting, every sampled path receives the same weight, so the target-prefix probability estimate has zero variance. At $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) = 0$ we set $\psi^*(\boldsymbol{x}) \stackrel{\text{def}}{=} \mathbb{1}\{\texttt{is_member}(\boldsymbol{x}, \boldsymbol{y})\}$, matching Equation (5.4) under the conventions of Section 2.2. The optimal twist returns the share of a prefix's mass that can still cover the target.

The next lemma gives the backward recursion for the optimal twist and its subtree-mass identity. The recursion mirrors prune-free `beam_summing`. Its cylinder and stop terms are exactly the contributions accumulated by

the algorithm, and unfolding the recursion from the empty prefix gives the quotient–remainder decomposition. Write $\mathcal{A}(\boldsymbol{x}) \stackrel{\text{def}}{=} \{x' \in \mathcal{X} : \texttt{is_live}(\boldsymbol{x}x', \boldsymbol{y})\}$ for the **live set** of next source symbols.

Lemma 5.1 (A backward recursion for the optimal twist)**.** *The optimal twist ψ^* of Equation (5.3) (relative to $\boldsymbol{y}$) satisfies the backward recursion*

$$\psi^*(\boldsymbol{x}) = \begin{cases} 1 & \textbf{\textit{if}}\ \texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y}) \\ \mathbb{1}\{\texttt{is_member}(\boldsymbol{x}, \boldsymbol{y})\}\, \overrightarrow{p_{\mathcal{X}}}(\textsc{eos} \mid \boldsymbol{x}) \\ \qquad + \sum_{x' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \psi^*(\boldsymbol{x}x') & \textbf{\textit{otherwise}} \end{cases} \tag{5.4}$$

and every source prefix $\boldsymbol{x}$ obeys

$$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\, \psi^*(\boldsymbol{x}) = \sum_{\substack{\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y}) \\ \boldsymbol{x} \preceq \boldsymbol{x}'}} p_{\mathcal{X}}(\boldsymbol{x}'), \tag{5.5}$$

the source mass of the completions of $\boldsymbol{x}$ that cover $\boldsymbol{y}$, and $\psi^*(\varepsilon) = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$.

Proof. Suppose first that $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) > 0$. If $\texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y})$ holds, every completion of $\boldsymbol{x}$ covers $\boldsymbol{y}$, so $\psi^*(\boldsymbol{x}) = 1$. Otherwise, condition on whether the completion stops immediately with EOS or begins with a source symbol x'. The stop event covers $\boldsymbol{y}$ exactly when $\texttt{is_member}(\boldsymbol{x}, \boldsymbol{y})$ holds, and after drawing x' the conditional probability of covering $\boldsymbol{y}$ is $\psi^*(\boldsymbol{x}x')$. A non-live child has optimal twist zero, so the sum may be restricted to $\mathcal{A}(\boldsymbol{x})$, giving Equation (5.4).

When $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) = 0$, the recursion follows from the conventions of Section 2.2 and the definition of ψ^* when $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) = 0$. If $\boldsymbol{x}$ is a cylinder, then it is a member and both sides of Equation (5.4) equal one. Otherwise, both sides equal $\mathbb{1}\{\texttt{is_member}(\boldsymbol{x}, \boldsymbol{y})\}$.

It remains to prove Equation (5.5). For $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) > 0$, the definition of the optimal twist gives

$$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\, \psi^*(\boldsymbol{x}) = \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \sum_{\boldsymbol{x}'' \in \mathcal{X}^*} \frac{p_{\mathcal{X}}(\boldsymbol{x}\boldsymbol{x}'')}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})} \mathbb{1}\{\texttt{is_member}(\boldsymbol{x}\boldsymbol{x}'', \boldsymbol{y})\} \tag{5.6}$$

$$= \sum_{\substack{\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y}) \\ \boldsymbol{x} \preceq \boldsymbol{x}'}} p_{\mathcal{X}}(\boldsymbol{x}'). \tag{5.7}$$

If $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) = 0$, every extension $\boldsymbol{x}' \succeq \boldsymbol{x}$ has $p_{\mathcal{X}}(\boldsymbol{x}') = 0$, so both sides of

Equation (5.5) are zero.

Finally, setting $\boldsymbol{x} = \varepsilon$ in Equation (5.5) gives

$$\psi^*(\varepsilon) = \sum_{\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}') = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}), \tag{5.8}$$

by Equation (3.2). ■

5.1.2 Stochastic pruning

Running `beam_summing` with either of the prune functions in Algorithm 5.1 can give a biased estimate. To make it unbiased, we introduce pruning functions that draw the survivors without replacement (SWOR) and correct for unequal selection using **Horvitz–Thompson reweighting** (Horvitz and Thompson, 1952). This preserves the pool's sums in expectation.

SWOR retains every positive-weight particle when there are at most M of them and otherwise replaces the pool by a size-M subset $\mathcal{S}$, reweighting survivors by their inclusion probabilities (Fearnhead and Clifford, 2003). In `beam_summing`, particle i in the pool passed to the prune is a child particle $\boldsymbol{x}^{(m)}x'$, with weight $w^{(i)} = w^{(m)}\,\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x}^{(m)})$ (Algorithm 5.1). For a function $h^{(i)} \stackrel{\text{def}}{=} h(\boldsymbol{x}^{(i)})$, SWOR estimates $\sum_{i=1}^{N} w^{(i)}h^{(i)}$ by

$$\widehat{\Sigma}_h \stackrel{\text{def}}{=} \sum_{i \in \mathcal{S}} \frac{w^{(i)}}{\pi_i}\, h^{(i)}. \tag{5.9}$$

The estimate is unbiased for any valid inclusion probabilities, those with $\pi_i = 0$ only where $w^{(i)}h^{(i)} = 0$ (Horvitz and Thompson, 1952). With $I_i \stackrel{\text{def}}{=} \mathbb{1}\{i \in \mathcal{S}\}$ and $\mathbb{E}_{\mathcal{S}}[I_i] = \pi_i$,

$$\mathop{\mathbb{E}}_{\mathcal{S}}[\widehat{\Sigma}_h] = \sum_{i=1}^{N} \mathop{\mathbb{E}}_{\mathcal{S}}[I_i]\, \frac{w^{(i)}}{\pi_i}\, h^{(i)} = \sum_{i=1}^{N} w^{(i)}\, h^{(i)}. \tag{5.10a}$$

Choosing the inclusion probabilities. First consider the untwisted case, $\psi \equiv 1$. Write $\mathcal{I}_+$ for the set of positive-weight child particles. When $|\mathcal{I}_+| \le M$ the prune keeps all of them unchanged. Otherwise, any valid probabilities give an unbiased estimate, and we set each child particle's inclusion probability proportional to its weight, capped at one,

$$\pi_i = \min\bigl(1,\, c\, w^{(i)}\bigr), \quad i = 1, \dots, N, \qquad \sum_{i=1}^{N} \min\bigl(1,\, c\, w^{(i)}\bigr) = M, \tag{5.11}$$

```
def swor_{M,ψ}(P'):
  W ← {w^(i) · ψ(x^(i))
     : (x^(i), w^(i)) ∈ P'}
  I_+ ← {i : W^(i) > 0}
  if |I_+| ≤ M:
    return {(x^(i), w^(i)) : i ∈ I_+}
  π ← pps(W, M)
  S ~ systematic(π, M)
  return {(x^(i), w^(i)/π_i) : i ∈ S}

def pps((w^(i))_{i=1}^N, M):
  K ← ∅
  π ← 0
  while True:
    π_i ← (M − |K|) w^(i) / Σ_{j∉K} w^(j)   (i ∉ K)
    K' ← {i ∉ K : π_i ≥ 1}
    if K' = ∅: break
    π_{K'} ← 1
    K ← K ∪ K'
  return π

def systematic(π, M):
  φ ~ uniform perm. of {1, ..., N}
  C_i ← Σ_{j≤i} π_{φ(j)}   (i = 0 ... N)
  u ~ Uniform(0, 1)
  return {φ(i) : ∃ k < M, C_{i−1} ≤ u + k < C_i}

def swor_adaptive_{M,ψ,ρ}(P', A):
  W^(i) ← w^(i) · ψ(x^(i))
  W ← Σ_i W^(i)
  I_+ ← {i : W^(i) > 0}
  if W = 0:
    return ∅
  μ ← Σ_i min(1, W^(i) M / (ρ (A + W)))
  m ← min(M, ⌊μ + 1/2⌋)
  if m = 0:
    B ~ Bernoulli(μ)
    if B = 0: return ∅
    m ← 1,  w^(i) ← w^(i)/μ   (∀i)
  if |I_+| ≤ m:
    return {(x^(i), w^(i)) : i ∈ I_+}
  π ← pps(W, m)
  S ~ systematic(π, m)
  return {(x^(i), w^(i)/π_i) : i ∈ S}

def beam_swor(y, M, ψ):
  return beam_summing(y, ψ, swor)

def beam_swor_adaptive(y, M, ψ):
  return
    beam_summing(y, ψ, swor_adaptive)
```

Algorithm 5.2: **SWOR.** The without-replacement resample `swor` replaces the `prune` in Algorithm 5.1, and `swor_adaptive` uses the mass accumulated so far to set the number of particles retained at each prune. To ensure halting even when $A = 0$, this calculation uses the effective value $A_{\epsilon,t}$ of Equation (5.20). Section 5.1.4 gives the full construction and its almost-sure halting guarantee. `beam_swor` and `beam_swor_adaptive` are `beam_summing` run with these prunes.

with the constant $c > 0$ fixed by the second equation. The capped child particles form the **certainty set** and survive with their weights unchanged (Shah and Kroese, 2018, §3.2.5), and every non-certain survivor carries the same reweighted value $w^{(i)}/\pi_i$. This means that the reweighted total matches the pool total on every draw, the randomness deciding which prefixes carry the mass forward.

The pruning function `swor` (Algorithm 5.2) combines these pieces. If the pool has more than M positive twisted weights, it hands them to `pps` (probabilities proportional to size) for the inclusion probabilities $\pi_i \propto w^{(i)}\,\psi(\boldsymbol{x}^{(i)})$ and their certainty set, and draws the size-M subset with `systematic` (Algorithm 5.2). Any fixed-size draw with those inclusion probabilities qualifies,

and `systematic` realizes them exactly in $\mathcal{O}(N)$ (Madow, 1949). Each survivor gets the Horvitz–Thompson weight $w^{(i)}/\pi_i$ of Equation (5.9).

The recursion in `pps` (Algorithm 5.2) finds the certainty sets, pins child particles whose inclusion probabilities reach one, and recurses on the others with the remaining sample size. Zero-weight child particles keep $\pi_i = 0$ and are never drawn. `beam_summing` with `swor` as its pruning function is the particle filter of Fearnhead and Clifford (2003), instantiated for the source-prefix enumeration over the precover. We write `beam_swor` for `beam_summing` run with the `swor` prune (Algorithm 5.2).

The second pruning function of Algorithm 5.2, `swor_adaptive`, uses the current-position running total to set the next live-particle count. We introduce the resulting `beam_swor_adaptive` algorithm and its almost-sure halting guarantee in Section 5.1.4, once the SWOR guarantees are in place (Section 5.1.3).

5.1.3 Unbiasedness guarantees

Without pruning, each particle carries its exact prefix mass (Section 5.1.1). The lemma and theorem below show that if each pruning step preserves these pool weights in expectation and the run halts, the accumulated quotient and remainder contributions form an unbiased estimate.

For a pool $\mathscr{P}$, write $w_{\mathscr{P}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{(\boldsymbol{x}',w)\in\mathscr{P}} w\,\mathbb{1}\{\boldsymbol{x}' = \boldsymbol{x}\}$ for its weight at a source prefix $\boldsymbol{x} \in \mathcal{X}^*$. For every fixed choice of the inputs to the prune, let $\widetilde{\mathscr{P}} \sim \texttt{prune}(\mathscr{P}', \cdots)$ denote its random output. We require

$$\mathop{\mathbb{E}}_{\widetilde{\mathscr{P}}\sim\texttt{prune}(\mathscr{P}',\cdots)}[w_{\widetilde{\mathscr{P}}}(\boldsymbol{x})] = w_{\mathscr{P}'}(\boldsymbol{x}), \qquad \text{for every } \boldsymbol{x} \in \mathcal{X}^*. \tag{5.12}$$

The expectation in Equation (5.12) is only over the prune's internal randomness, with all its inputs, including $\mathscr{P}'$ and A, held fixed.

For a run of `beam_summing` on a target string $\boldsymbol{y}$, consider the call for $\boldsymbol{y}$, at the top of the recursion. Its starting pool $\mathscr{P}_1$ holds the re-queued quotient entries of the target one symbol shorter, with total weight one at ε when $\boldsymbol{y} = \varepsilon$. For $i \geq 1$, the extension step builds $\mathscr{P}'_i$ from $\mathscr{P}_i$, and the prune returns $\mathscr{P}_{i+1} = \texttt{prune}(\mathscr{P}'_i, \cdots)$. Write $\overline{\mathscr{P}}_i$ for the formal pools generated by the corresponding prune-free recursion, defined through their weight at each source prefix even if the complete reference enumeration does not halt.

Lemma 5.2 (Pruning preserves the unpruned pool weights in expectation)**.** *Fix a target string $\boldsymbol{y}$ and run* `beam_summing` *(Algorithm 5.1), with some prune function satisfying Equation* (5.12)*. Then, at every iteration*

i and every source prefix $\boldsymbol{x}$,

$$\mathbb{E}[w_{\mathscr{P}_i}(\boldsymbol{x})] = w_{\overline{\mathscr{P}}_i}(\boldsymbol{x}). \tag{5.13}$$

Proof. We induct on $N = |\boldsymbol{y}|$. For each N, we first establish the claim for the initial pools at $i = 1$, then induct on the iterations within that call.

Initial pools. For $N = 0$, $\boldsymbol{y} = \varepsilon$, both runs start with aggregate weight one at the empty prefix and zero at every other prefix. Now let $N > 0$ and assume the result for the recursive call on $\boldsymbol{y}_{<N}$. The pool $\mathscr{P}_1$ is exactly the quotient returned by the recursive call on $\boldsymbol{y}_{<N}$, while $\overline{\mathscr{P}}_1$ contains the corresponding prune-free quotient weights. Fix a source prefix $\boldsymbol{x}$. Because the starting pool is prefix-free, $\boldsymbol{x}$ can extend at most one of its entries. Since each iteration appends one source symbol, $\boldsymbol{x}$ can be added to the call's quotient at most once. The outer induction hypothesis therefore implies that the quotient has the same expected weight at every source prefix as its prune-free counterpart, so

$$\mathbb{E}[w_{\mathscr{P}_1}(\boldsymbol{x})] = w_{\overline{\mathscr{P}}_1}(\boldsymbol{x}). \tag{5.14}$$

Together with the case $N = 0$, this establishes the within-call induction base $i = 1$ for every N.

Within-call induction. Fix N and assume the identity holds at iteration i for every source prefix. The full extension step obeys

$$w_{\mathscr{P}'_i}(\boldsymbol{x}x') = w_{\mathscr{P}_i}(\boldsymbol{x})\, \mathbb{1}\{\neg \texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y})\}\, \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \mathbb{1}\{\psi(\boldsymbol{x}x') > 0\}. \tag{5.15}$$

A cylinder prefix produces no children, while every other particle produces one child for each symbol admitted by the twist. For every prefix $\boldsymbol{x}$ and symbol x', we therefore have

$$\begin{aligned}
\mathbb{E}[w_{\mathscr{P}_{i+1}}(\boldsymbol{x}x')] &= \mathbb{E}[w_{\mathscr{P}'_i}(\boldsymbol{x}x')] && \text{(5.16a, cond. on prune and Equation (5.12))} \\
&= \mathbb{E}[w_{\mathscr{P}_i}(\boldsymbol{x})]\, \mathbb{1}\{\neg \texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y})\} && \text{(5.16b)} \\
&\qquad \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \mathbb{1}\{\psi(\boldsymbol{x}x') > 0\} && \text{(5.16c, Equation (5.15))} \\
&= w_{\overline{\mathscr{P}}_i}(\boldsymbol{x})\, \mathbb{1}\{\neg \texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y})\}\, \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \mathbb{1}\{\psi(\boldsymbol{x}x') > 0\} \\
&\qquad && \text{(5.16d, induction hypothesis)} \\
&= w_{\overline{\mathscr{P}}'_i}(\boldsymbol{x}x') && \text{(5.16e, Equation (5.15) in the reference run)} \\
&= w_{\overline{\mathscr{P}}_{i+1}}(\boldsymbol{x}x'). && \text{(5.16f, } \overline{\mathscr{P}}_{i+1} = \overline{\mathscr{P}}'_i \text{ under the identity prune)}
\end{aligned}$$

Since each iteration appends one source symbol, every prefix in either next pool is nonempty and has the form $\boldsymbol{x}x'$. This proves the claim at iteration $i+1$ for every source prefix. Thus the within-call induction proves the claim at every iteration, completing the outer induction on N. ■

Theorem 5.1 (Unbiasedness)**.** *Fix a nonempty target string $\boldsymbol{y}$ and run* `beam_summing` *(Algorithm 5.1) with a twist (Section 5.1.1). Suppose that*
(1) every prune preserves the pool's expected weight at every prefix (Equation (5.12)), and
(2) the run halts almost surely.
Then $\mathbb{E}[\hat{Z}_{|\boldsymbol{y}|}] = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$.

Proof. By hypothesis (2), every recursive call returns almost surely, so the contributions V_t below and the final estimator $\hat{Z}_{|\boldsymbol{y}|}$ are well defined. For each $1 \le t \le |\boldsymbol{y}|$ and each source prefix $\boldsymbol{x}$, let $\overline{V}_t(\boldsymbol{x})$ be the contribution assigned to $\boldsymbol{x}$ by the deterministic prune-free reference enumeration at position t, as in Equation (5.1). The twist only gates the extension step in `beam_summing`. By definition it gates off no child source prefix with positive covering mass and does not alter particle weights. The exactness of prune-free `beam_summing` (Section 5.1.1) therefore gives $\overline{V}_t(\boldsymbol{x}) = \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}_{\le t})$, while $\overline{V}_t(\boldsymbol{x})\overrightarrow{p_{\mathcal{X}}}(\textsc{eos} \mid \boldsymbol{x}) = p_{\mathcal{X}}(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y}_{\le t})$.

We first show that pruning preserves these contributions in expectation, $\mathbb{E}[V_t(\boldsymbol{x})] = \overline{V}_t(\boldsymbol{x})$ for every $1 \le t \le |\boldsymbol{y}|$ and every $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}_{\le t}) \sqcup \mathcal{R}(\boldsymbol{y}_{\le t})$, by induction on t. Fix such a position t and an element $\boldsymbol{x}$ of its decomposition. Since $\mathcal{P}(\boldsymbol{y}_{\le t}) \subseteq \mathcal{P}(\boldsymbol{y}_{\le t-1})$, the element also belongs to the preceding precover. The preceding quotient–remainder split gives two alternatives. Either $\boldsymbol{x}$ extends a unique $\boldsymbol{x}' \in \mathcal{Q}(\boldsymbol{y}_{\le t-1})$, or $\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y}_{\le t-1})$ (Chapter 3). When $t = 1$, $\mathcal{Q}(\varepsilon) = \{\varepsilon\}$ and $\mathcal{R}(\varepsilon) = \emptyset$, so only the quotient alternative occurs. When $t > 1$, assume the identity holds at position $t-1$. If $\boldsymbol{x}$ is a remainder element at position t and $\overrightarrow{p_{\mathcal{X}}}(\textsc{eos} \mid \boldsymbol{x}) = 0$, the convention in Equation (5.1) gives $V_t(\boldsymbol{x}) = \overline{V}_t(\boldsymbol{x}) = 0$. Thus, if $\boldsymbol{x}$ is a remainder element, only the case $\overrightarrow{p_{\mathcal{X}}}(\textsc{eos} \mid \boldsymbol{x}) > 0$ remains.

Extended from the quotient. Let $\boldsymbol{x}'$ denote the quotient prefix in the first alternative. The call at t places the preceding quotient entries at $\boldsymbol{x}'$ in $\mathscr{P}_1$ and appends one source symbol per iteration. Thus any entries at $\boldsymbol{x}$ can occur only at iteration $i = |\boldsymbol{x}| - |\boldsymbol{x}'| + 1$. At that iteration, Equation (5.1) identifies the contribution assigned to $\boldsymbol{x}$ with its aggregate pool weight,

so $V_t(\boldsymbol{x}) = w_{\mathscr{P}_i}(\boldsymbol{x})$ and $\overline{V}_t(\boldsymbol{x}) = w_{\overline{\mathscr{P}}_i}(\boldsymbol{x})$. Applying Lemma 5.2 gives

$$\mathbb{E}[V_t(\boldsymbol{x})] = \mathbb{E}[w_{\mathscr{P}_i}(\boldsymbol{x})] = w_{\overline{\mathscr{P}}_i}(\boldsymbol{x}) = \overline{V}_t(\boldsymbol{x}).$$

Retained from the remainder. In the second alternative, the target re-check passes every entry at $\boldsymbol{x}$ unchanged because $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y}_{\leq t})$ (Section 5.1.1). Hence $V_t(\boldsymbol{x}) = V_{t-1}(\boldsymbol{x})$ and $\overline{V}_t(\boldsymbol{x}) = \overline{V}_{t-1}(\boldsymbol{x})$. The induction hypothesis therefore gives

$$\mathbb{E}[V_t(\boldsymbol{x})] = \mathbb{E}[V_{t-1}(\boldsymbol{x})] = \overline{V}_{t-1}(\boldsymbol{x}) = \overline{V}_t(\boldsymbol{x}).$$

The zero-EOS-probability observation and the two alternatives complete the induction. Set $T = |\boldsymbol{y}|$. Then

$$\begin{aligned}
\mathbb{E}[\hat{Z}_T] &= \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} \mathbb{E}[V_T(\boldsymbol{x})]\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \;+ \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \mathbb{E}[V_T(\boldsymbol{x})] \\
&\qquad\qquad \text{(5.17a, Equation (5.2) and Tonelli)} \\
&= \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} \overline{V}_T(\boldsymbol{x})\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \;+ \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overline{V}_T(\boldsymbol{x}) \\
&\qquad\qquad \text{(5.17b, the preservation above)} \\
&= \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \;+ \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \quad \text{(5.17c, the prune-free identities above)} \\
&= \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}). \qquad\qquad \text{(5.17d, Equation (3.6))}
\end{aligned}$$

■

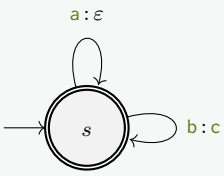
The stochastic pruning functions we consider (Algorithm 5.2) meet the expected-weight hypothesis of Theorem 5.1, while halting is established separately for each algorithm. The draw `swor` meets hypothesis (1) by the Horvitz–Thompson identity (Equation (5.9)), its inclusion probabilities are positive wherever the twisted weight is, and the extension step only hands it particles with positive twist (Section 5.1.1). For hypothesis (2), the full `beam_swor` run halts whenever the decomposition is finite at every recursive prefix position. For target-prefix calls whose source-prefix enumeration would otherwise continue indefinitely, the tail-roulette prune of Section B.1.4 preserves hypothesis (1) and makes each call halt almost surely. For `beam_swor_adaptive`, both hypotheses follow from Proposition 5.1.

5.1.4 Adaptive-budget SWOR and almost-sure halting

For a finite quotient–remainder decomposition, every branch of the enumeration in `beam_summing` reaches a cylinder or a dead end (a prefix whose live set $\mathcal{A}(\boldsymbol{x})$ is empty) after finitely many iterations, so the call halts (Theorem 4.1). When the decomposition is infinite, the call need not halt.

Example 5.1 — An infinite quotient

Consider the one-state transducer below and the target $\boldsymbol{y} = \mathsf{c}$. Both transitions loop on the single initial and final state. The source symbol a emits nothing, while b emits c, so a source string maps to one c for each b it contains.



Choose a source model that assigns positive probability to both a and b after every prefix a^k. A prefix covers c once it holds a b, and every extension keeps covering, so the decomposition consists entirely of the quotient, with one cylinder at every depth.

$$\mathcal{Q}(\mathsf{c}) = \{\mathsf{a}^k\mathsf{b} : k \geq 0\}, \qquad \mathcal{R}(\mathsf{c}) = \emptyset.$$

At depth k, the prefix a^k has two children, the cylinder child $\mathsf{a}^k\mathsf{b}$ and the prefix a^{k+1}. At the next iteration, $\mathsf{a}^k\mathsf{b}$ is recognized as a cylinder and its weight is added to A. The prefix a^{k+1} remains live, is never a member, a cylinder, or dead, and is extended in turn. For any $M \geq 2$, fixed-M SWOR keeps both child particles and no randomness enters, so the call never halts. Although the weight along this path decays to zero, the source-prefix enumeration may continue forever.

To avoid the nontermination in Example 5.1, the fixed-M methods of Table 5.1 compose their usual prune with the tail-roulette prune of Section B.1.4. The prune presented below instead halts this example on its own.

The `swor` function uses a fixed cap of M particles at every iteration. We instead decrease the live-particle count as the current-position running total grows, saving compute. The result is the `swor_adaptive` prune of Algorithm 5.2, and `beam_swor_adaptive` is `beam_summing` run with it. We call

this method **adaptive-budget SWOR**. At every `prune` call (Section 5.1.2 of Algorithm 5.2), we set

$$c = \frac{M}{\rho \cdot (A + W)}. \tag{5.18}$$

Here A is the current-position running total (Algorithm 5.1), W is the pool's total twisted weight, and M is the maximum particle count. The corresponding per-particle threshold $1/c$ sets the twisted-weight scale for the count μ. Thus $\rho \in (0, 1]$ controls this count and, through m, the number of particles retained. Children below the threshold contribute in proportion to their twisted weight, while those at or above it contribute one (the dashed line of Figure 5.1(a,b)). The resulting μ fixes the particle count m before sampling according to

$$\mu = \sum_{i=1}^{N} \min\Big(1,\ cw^{(i)}\psi(\boldsymbol{x}^{(i)})\Big), \qquad m = \min\big(M,\ \lfloor \mu + \tfrac{1}{2} \rfloor\big). \tag{5.19}$$

This replaces the calculation of Equation (5.11), where the constant is solved so that the capped counts sum to the fixed particle count M. `pps` then keeps exactly m particles, re-solving the constant to c', so every non-pinned survivor carries the shared weight $1/c'$ (Figure 5.1(c)). In the limit $\rho \to 0$, `swor_adaptive` recovers the fixed-M `swor` prune. Under the live twist, the capped count satisfies $\mu \leq (M/\rho)\, W/(A+W)$, where $W/(A+W)$ is the pool's remaining live share. A source-extension iteration never increases the pool's mass. When the pool counts for less than half a particle, $\mu < \frac{1}{2}$, a Russian roulette draw (Kahn, 1955) decides whether to empty the pool (Section 5.1.2). To also cover a position that can continue forever without adding anything to A, we replace it at position t by

$$A_{\epsilon,t} \stackrel{\text{def}}{=} \max\{A, \epsilon \hat{Z}_{t-1}\}, \qquad \hat{Z}_0 \stackrel{\text{def}}{=} 1, \quad \epsilon > 0. \tag{5.20}$$

This substitution applies only to the count and threshold in Section 5.1.2 and Equation (5.18). The capped count μ therefore vanishes as the pool's total live weight W vanishes relative to $A_{\epsilon,t}$. The preceding estimate $\hat{Z}_{t-1}$ is fixed before position t begins. The resulting particle count is therefore fixed before sampling, so the SWOR draw still preserves each weight in expectation. Figure 5.1 traces one draw through panels (a) to (c), as A rises and the particle count m falls.

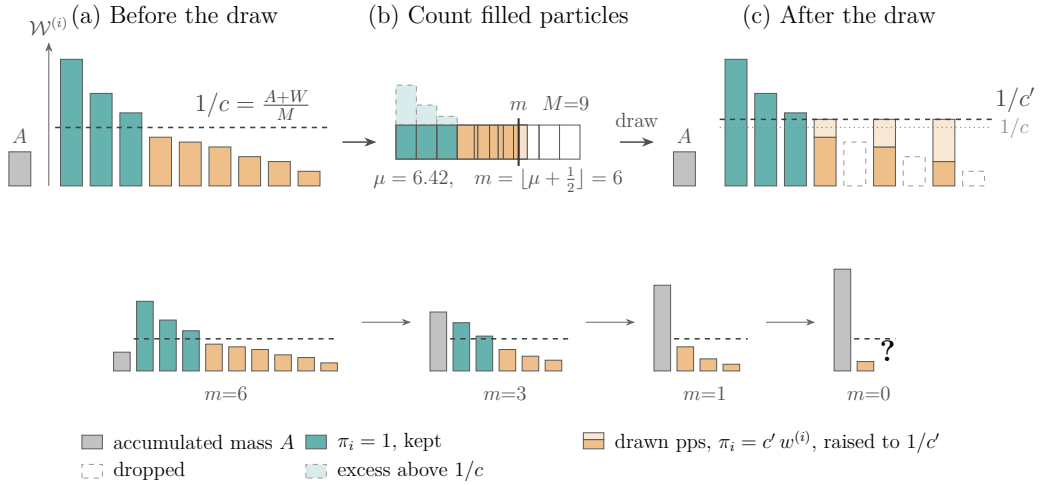


Figure 5.1: One `swor_adaptive` prune, left to right (top), and successive prunes at one target position (bottom). Each bar is one child particle of the pool with height its weight. The dashed line is the per-particle threshold $1/c = \rho\,(A + W)/M$ (drawn at ρ=1). Panel (b) shows how each child particle fills $\min(c\,w^{(i)}, 1)$ of one of the M slots, so a child particle above the threshold contributes at most $1/c$ to the capped mass used for the count, regardless of its weight, the counts sum to μ, and the particle count $m = \lfloor \mu + \frac{1}{2} \rfloor$, the nearest integer, is fixed before any randomness. In panel (c), `pps` then keeps exactly m particles, re-solving the constant to c', pinning the child particles above the new threshold, and drawing each remaining survivor with probability $c'\,w^{(i)}$ at the shared weight $1/c'$, so the draw passes the twisted total through unchanged (Proposition 5.1). Bottom row: as A grows, the pool's bars shrink against the threshold and the particle count falls. When the pool counts for less than half a particle, a coin flip determines whether the call continues with a one-particle draw or halts.

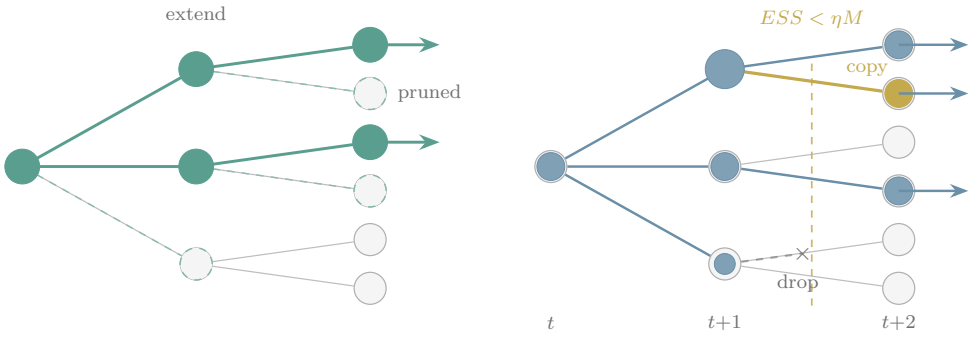
Proposition 5.1 (Adaptive-budget SWOR)**.** *Fix a positive integer M, a twist ψ, $\rho \in (0, 1]$, and $\epsilon > 0$, and run* `beam_swor_adaptive` *using $A_{\epsilon,t}$ from Equation* (5.20)*. Then (I) each prune call satisfies Equation* (5.12)*. Under the live twist $\psi =$* `is_live`*, (II) the full run halts almost surely. Consequently, under the live twist, $\mathbb{E}[\hat{Z}_t] = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}_{\leq t})$ at every position.*

Proof. The proof is deferred to Section B.1.5. ■

Figure 5.5 compares the method's accuracy and walltime, and Section B.2 gives an ablation of ρ.

5.1.5 SMC with multinomial resampling

The `beam_swor` estimator (Algorithm 5.2) keeps the `beam_summing` enumeration and randomizes only the pruning. A standard sequential Monte Carlo (SMC) construction instead uses multinomial resampling with replacement, gated by an *effective sample size* test (ESS, Section B.1), as its prune (Figure 5.2). Two such algorithms appear in Table 5.1 and serve as the baselines in Section 5.2. First, `smc_simple` (Algorithm B.1) is a textbook particle filter (Doucet, Freitas, and Gordon, 2001; Naesseth, Lindsten, and Schön, 2019) run with the precover-restricted proposal. It collects mass only when a particle happens to draw EOS at a member prefix, and its resample duplicates heavy particles, so the pool's M slots hold dependent repeats exactly where the weights have collapsed. Second, `smc_rb` (Algorithm B.2) marginalizes the stop branch rather than drawing it, a Rao–Blackwellization of the stop decision. Both `smc_rb` and `beam_swor` use importance weighting to account for the probability mass of children left out of the pool. `smc_rb` samples one live child for each particle and may resample particles with replacement. `beam_swor` instead forms every live child before pruning and samples distinct survivors without replacement. Section B.1 develops both SMC baselines in full. It proves `smc_rb` unbiased with the same nested induction as Section 5.1.3, after showing that its sampled extension gives each child the same expected weight as the all-child extension.



(a) All-child extension followed by SWOR pruning (`beam_swor`, `beam_swor_adaptive`).

(b) Sampled extension followed by ESS-triggered multinomial resampling (`smc_simple`, `smc_rb`).

Figure 5.2: Two ways to maintain a bounded particle pool. Dashed rings in (a) mark children removed by pruning. In (b), the cross marks a dropped particle and orange marks a duplicate. Gray edges show unexplored branches. Dot size shows weight.

5.1.6 Relation to prior work

Our SWOR prune functions combine unequal-probability sampling without replacement with Horvitz–Thompson reweighting (Horvitz and Thompson, 1952). We implement the draw using systematic pps (Madow, 1949) and the certainty-set construction from without-replacement particle-filter resampling (Fearnhead and Clifford, 2003). Conditional Poisson stochastic beam search (Meister et al., 2021a) likewise samples prefixes without replacement, but its Horvitz–Thompson estimate of expectations over complete source strings requires their inclusion probabilities under the full search. We instead correct each pruning draw locally, and these expected-weight identities compose into an unbiased estimate of the precover's total mass (Shah and Kroese, 2018). Shah and Kroese (2018) consider a fixed number of sampling stages, whereas a target-prefix call may require an unbounded number of source-extension iterations. We extend the argument to this setting while adding quotient and remainder contributions, recursing over target positions, and adapting the particle count (Theorem 5.1 and Proposition 5.1).

5.2 Experiments

On paragraphs from WikiText-2 (Merity et al., 2017), we measure the bias of threshold-pruned beam summing and the variance of the unbiased estimates across particle counts (Section 5.2.1). Section 5.2.2 compares running time and variance. On a DNA-to-amino-acid transduction whose precover grows exponentially, Section 5.2.3 shows that `beam_summing` run with $\texttt{prune}_\tau$ gets prohibitively slow after a dozen amino acids, while `beam_swor_adaptive` reaches position 200 in about five seconds. Chapter 6 applies `swor_adaptive` to the PTB-word surprisal analysis (Section 6.5).

Estimators. We evaluate the four unbiased methods `smc_simple`, `smc_rb`, `beam_swor`, and `beam_swor_adaptive` (Algorithm 5.2 and Section 5.1.5), against the beam summing of Chapter 4, `beam_summing` with the mass prune $\texttt{prune}_\tau$ (Algorithm 5.1). Every setting uses the live twist `is_live`. The comparison in Section 5.2.4 separates the effect of marginalizing the EOS decision rather than sampling it (`smc_simple` versus `smc_rb`) from the combined change to extending every live child and pruning without replacement (`smc_rb` versus `beam_swor`).

Reporting. Over S seeds we report the log of the averaged estimate, $\log\left(\frac{1}{S}\sum_{s=1}^{S}\hat{Z}_s\right)$. Each seed's $\hat{Z}_s$ is unbiased for Z, so the average is unbiased

as well, and its log estimates $\log Z$. Averaging in the other order, $\frac{1}{S}\sum_s \log \hat{Z}_s$, would estimate $\mathbb{E}[\log \hat{Z}]$ instead, and log is concave, so by Jensen's inequality $\mathbb{E}[\log \hat{Z}] \leq \log \mathbb{E}[\hat{Z}] = \log Z$, a downward bias. We report between-seed standard deviations alongside method comparisons. We report particle counts as the mean number of live particles.

5.2.1 Byte-to-word transduction on real text

The PTB transducer used in Chapters 4 and 6, denoted with f_{ptb}, encodes the PTB tokenization (Marcus, Santorini, and Marcinkiewicz, 1993). We evaluate it on the same ten WikiText-2 paragraphs as in Chapter 4. We now measure what the bias correction changes in this setting using a bigram model. The source model p_{bigram} is a smoothed byte-level bigram model trained on the WikiText-2-raw train split and evaluated on the test paragraphs.[3]

We calculate the exact target prefix probabilities by composing f_{ptb} with the weighted finite-state representation of p_{bigram}. Standard forward and backward path sums in the resulting automaton give the per-position values, as described in Section B.3.

Results are shown in Figure 5.3, using the relative root-mean-square error across seeds, $\text{RMSE}[\hat{Z}/Z] = \sqrt{S^{-1}\sum_{s=1}^{S}(\hat{Z}_s/Z - 1)^2}$, as a function of the mean live particle count. On the single paragraph, threshold-pruned beam summing has error floors from 0.48 at the lowest threshold to 1.0 at the harshest. On the corpus, its error remains close to 1. The with-replacement samplers are unbiased, but their corpus error remains near 1 even with thousands of particles. `beam_swor` and `beam_swor_adaptive` drive the corpus error down to 0.17 and 0.21 at their largest tested M values, between-seed standard deviations 57× and 49× below `smc_simple`'s (Table 5.2). Panel (c) shows they resolve most paragraphs where `smc_rb` needs M in the thousands to do so and `smc_simple` never resolves the full corpus. Section B.5 shows the comparison at every position of all ten paragraphs. The two SWOR methods track the exact ratio closely, while `beam_summing` with $\texttt{prune}_\tau$ falls below it when pruning discards covering mass.

GPT-2 on PTB. We also run f_{ptb} with GPT-2 Large, p_{gpt2} (Radford et al., 2019). Exact target prefix probabilities are unavailable for GPT-2 on the WikiText paragraphs, so we report between-seed variance and convergence as M grows and compare with the threshold-pruned lower bound. The estimates in Figure 5.4 are shown for `smc_rb`, the lowest-variance method with

[3]We use add-α smoothing with $\alpha = 0.5$ (Jurafsky and Martin, 2026, Ch. 3).

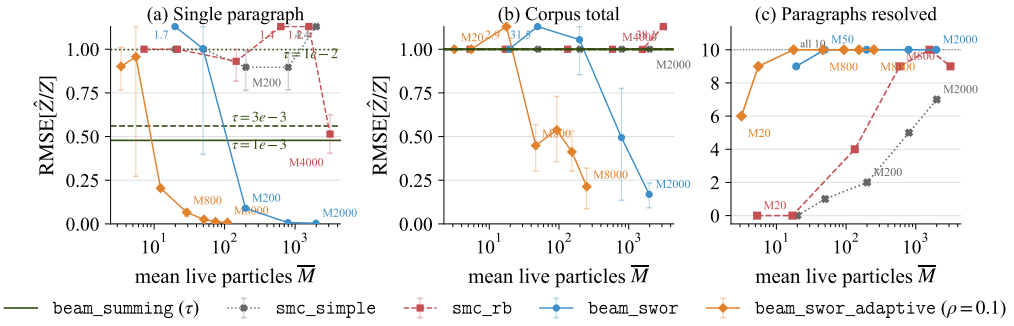


Figure 5.3: Unbiased methods versus beam summing with $\texttt{prune}_\tau$ on the $p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$ TLM, compared with exact probabilities. Panels (a) and (b) plot relative root-mean-square error RMSE$[\hat{Z}/Z]$: (a) a single example paragraph and (b) the ten-paragraph corpus total, both against the mean live particle count. Labels give the maximum particle count M. Error bars give 95% bootstrap intervals. Panel (c) counts the paragraphs each method resolves, capturing at least 5% of a paragraph's mass.

replacement, and `beam_swor_adaptive`, the lowest-variance method without replacement (Table 5.2), along with the lower bound at $\tau = 10^{-3}$. At the largest tested value of M, the two agree within 0.5 nats on the paragraph. On the corpus, `smc_rb` remains 4.7 nats below the `beam_swor_adaptive` plateau, which is reached at a much smaller mean live particle count. Table B.1 reports estimates for each of the ten paragraphs. The threshold-pruned estimate lies below the `beam_swor_adaptive` estimate on every paragraph, by 0.24 to 7.7 nats and 33.4 nats in total. The between-seed standard deviation of `beam_swor_adaptive` is smaller than this gap on every paragraph and at least an order of magnitude smaller on five of ten.

5.2.2 Comparing compute–variance frontiers

We compare the estimators against walltime in Figure 5.5.[4] With the p_{bigram} source, there is no GPU overhead, and with p_{gpt2} the walltime includes the source model's forward passes. On $p_{\mathsf{gpt2}} \circ \mathsf{f}_{\mathrm{ptb}}$, `beam_swor_adaptive` at nominal M=200 takes 772 seconds, compared with 1224 seconds for `beam_summing` with $\texttt{prune}_\tau$. The M=800 point of Table B.1 takes 1.5× as long as that baseline. Under both source models, the `beam_swor_adaptive` frontier dominates both

[4]All timings come from one workstation with an Intel Core i9-12900KF, 128 GB of RAM, and a single NVIDIA RTX 3090 with 24 GB of memory. For the CPU experiments, we report the total CPU time of each run, pinned to a P-core on an otherwise idle machine. For GPT-2 we report wall-clock time with one job at a time on the GPU.

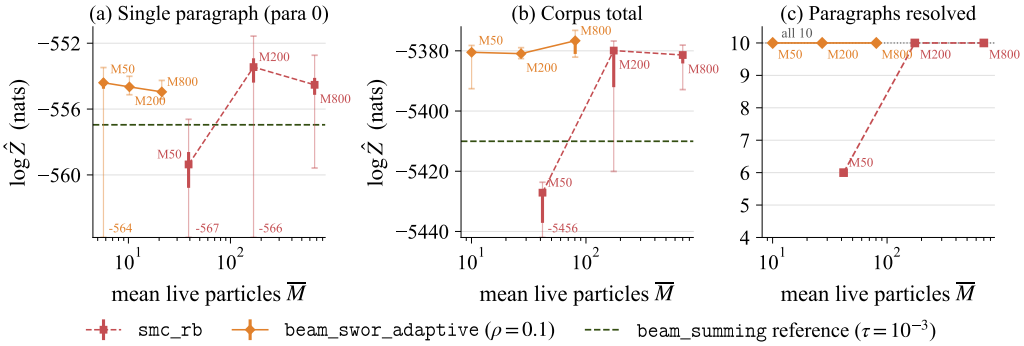
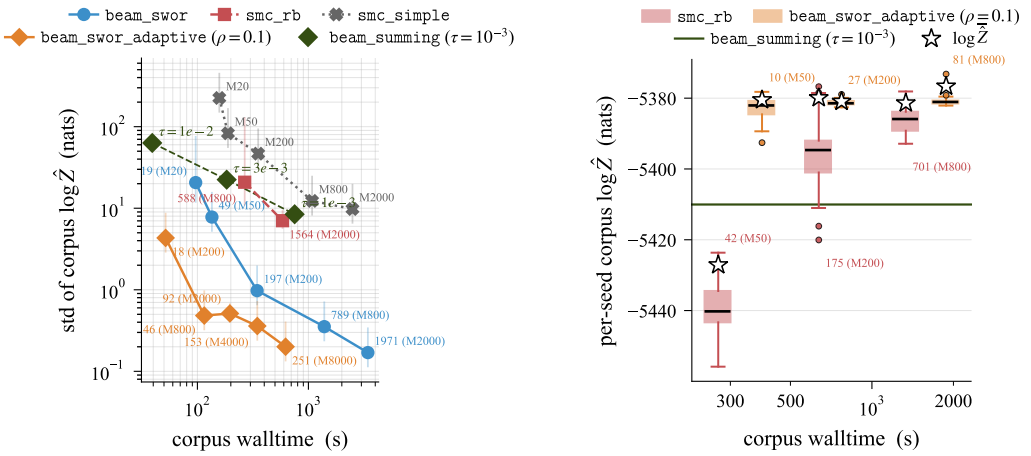


Figure 5.4: GPT-2 Large transduced with $\mathtt{f}_{\text{ptb}}$. Panels (a) and (b) plot the estimates $\log \hat{Z}$ in nats against the mean live particle count. Labels give the maximum particle count M. Each point is $\log\big(8^{-1}\sum_{s=1}^{8}\hat{Z}_s\big)$, with a failed run contributing $\hat{Z}_s = 0$. Thick bars give 95% bootstrap intervals, thin whiskers span the finite per-run log estimates, and clipped minima are printed at the lower edge. The dashed line is the threshold-pruned lower bound at $\tau{=}10^{-3}$. The two algorithms appear to converge to a common value. The lower bound is 2.0 nats below this value on the paragraph and approximately 33 nats below it on the corpus. Panel (c) counts the paragraphs on which each method recovers at least the lower bound's mass.

SMC baselines. On the bigram, at effectively matched between-seed standard deviation (0.36 against 0.35 nats), `beam_swor_adaptive` is four times faster than `beam_swor` (346 against 1390 seconds) and uses fewer live particles (153 against 789). At matched walltime, its standard deviation is nearly three times lower (0.36 against 0.98 nats). With $\rho{=}0.1$ at nominal $M{=}2000$, 149 seeds give a standard deviation of 0.163 nats (95% CI $[0.146, 0.184]$), consistent with the 0.170 nats of fixed-M `beam_swor` at about one fifth of its CPU time (Section B.2). The parameter ρ of `beam_swor_adaptive` is swept in Section B.2.

5.2.3 Transducing DNA language models

The DNA→amino-acid map f_{dna2aa} maps DNA strings over a four-letter alphabet to amino-acid strings. Combined with a high-entropy source model, it gives the setting where materializing the precover is most costly in our experiments. Most amino acids are the image of several source codons, and the high-entropy model places non-negligible mass on all of them, so the set of source strings an exhaustive method must track can multiply with each added symbol. We train a bigram model on the DNA training corpus, and use p_{dna}, a GPT-2 model trained on human DNA, the same model used in

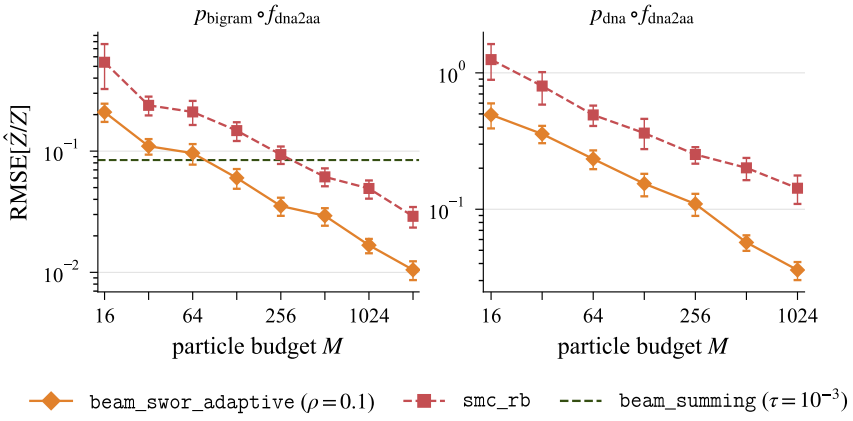


(a) p_{bigram} (b) p_{gpt2}

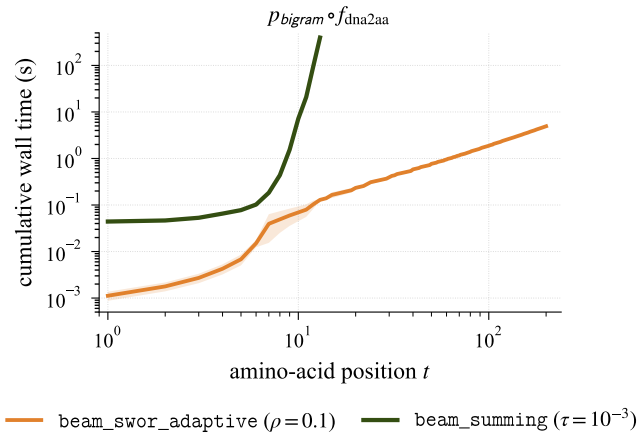
Figure 5.5: Accuracy–walltime comparisons. (a) For p_{bigram}, where exact target prefix probabilities are available, the between-seed standard deviation of the corpus log-estimate against per-seed corpus walltime. Annotations in both panels give the mean live particle count, with the maximum particle count M in parentheses. The `beam_summing` lines using $\texttt{prune}_\tau$ report bias relative to the exact values, and error bars give the 95% confidence interval of each standard deviation. (b) For p_{gpt2}, where exact target prefix probabilities are unavailable, the finite per-seed corpus log-estimates over 32 seeds. Boxes span the interquartile range with whiskers at 1.5 IQR and seeds beyond them drawn as dots, and the black bar marks the median seed. The star marks the pooled estimate $\log \bar{\hat{Z}}$, the log of the seed-averaged $\hat{Z}$, computed from all 32 seeds with a failed run contributing $\hat{Z} = 0$. Because the pooled estimate averages $\hat{Z}$ before taking the logarithm, occasional high estimates can make it larger than the median per-seed estimate. The `beam_summing` line shows its corpus total using $\texttt{prune}_\tau$.

Section 4.4. Figure 5.6a compares both DNA models with exact target prefix probabilities obtained by brute-force enumeration of the precover. In both the validation and scaling comparisons, `beam_summing` uses $\texttt{prune}_\tau$ with $\tau = 10^{-3}$.

Figure 5.6b shows the running time of `beam_summing` with $\texttt{prune}_\tau$ growing rapidly within a dozen amino acids, while `beam_swor_adaptive` scales approximately linearly. Section B.4 gives a uniform-source worst case for `beam_summing` with $\texttt{prune}_M$. Its captured mass ratio vanishes exponentially in the target length, while `smc_simple`, `smc_rb`, and fixed-M `beam_swor` are exact for any M.



(a) Validation against exact values.



(b) Cumulative wall time by amino-acid position.

Figure 5.6: DNA-to-amino-acid results. (a) Relative RMSE against exact values over 64 seeds and three targets, with 95% bootstrap intervals. The error of `beam_swor_adaptive` decreases faster with M than that of `smc_rb`, while `beam_summing` with `prune`$_\tau$ has fixed error. (b) Cumulative walltime for the bigram model. `beam_summing` with `prune`$_\tau$ exceeds five minutes at position 13, while `beam_swor_adaptive` (ρ=0.1, M=256) reaches position 200 in about five seconds.

5.2.4 Variance across estimators

Table 5.2 compares the four unbiased estimators across the three settings. Marginalizing the EOS decision rather than sampling it (`smc_simple` versus `smc_rb`) reduces variance only modestly. The large reduction in every setting occurs between `smc_rb` and `beam_swor`, which changes both the extension to every live child and the pruning to sampling without replacement. Compared to `beam_swor`, `beam_swor_adaptive` achieves comparable variance with less computation on the bigram corpus, lower variance on GPT-2, and identical results on DNA, where `swor_adaptive` retains the same number of particles as `swor` (Table 5.2 and Figure 5.5).

5.3 Summary

This chapter framed the prefix probability of a transduced language model as a normalizing constant and developed unbiased stochastic estimators for it. The SWOR estimators replace threshold pruning with reweighted sampling without replacement, while the adaptive variant saves computation by reducing the particle count as mass is accumulated. The algorithms apply beyond the transducers tested here whenever their live, member, and cylinder checks can be computed. We implement these checks with the lazily determinized frontier machinery of Chapter 4. Extending the stochastic method from prefix probabilities along a given target sequence to the full next-symbol distribution remains future work.

Table 5.2: Between-seed standard deviation, in nats, of each estimator's log probability estimate $\log \hat{Z}$. The columns report the ten-paragraph WikiText-2 total for the bigram model, the first WikiText-2 paragraph for GPT-2, and the amino-acid target for DNA. Reported values use eight seeds and maximum particle counts of 2000, 200, and 256, respectively, except `beam_swor_adaptive` on the bigram (M=8000, mean live count 251). The `smc_simple` GPT-2 run covered only 120/850 bytes.

Algorithm	$p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$	$p_{\mathsf{gpt2}} \circ \mathsf{f}_{\mathrm{ptb}}$	$p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{dna2aa}}$
`smc_simple`	9.78	—	0.111
`smc_rb`	7.86	3.23	0.107
`beam_swor`	0.170	0.395	0.062
`beam_swor_adaptive` (ρ=0.1)	0.200	0.199	0.062

Chapter 6

Surprisal in psycholinguistics

This chapter contains material reproduced and adapted from Kiegeland et al. (2026), with editorial changes for narrative consistency. The reanalysis in Section 6.5 contains material reproduced and adapted from Snæbjarnarson et al. (2026b).

A long line of work in psycholinguistics has sought to characterize the processing difficulty that a comprehender experiences upon encountering a linguistic unit in context (Balota, Pollatsek, and Rayner, 1985; Ehrlich and Rayner, 1981; Miller and McKean, 1964, *inter alia*). A prominent *computational* (Marr, 1982) account of such processing difficulty is surprisal theory (Hale, 2001; Levy, 2008), which posits that processing effort is determined by a unit's surprisal: the negative log-probability of encountering that unit given its preceding context.[1]

In early experimental work on surprisal theory, researchers often built and trained their own language models for a given dataset and experimental paradigm (Demberg and Keller, 2008; Goodkind and Bicknell, 2018; Hale, 2001; Levy, 2008; Mitchell et al., 2010, *inter alia*). Because they controlled the entire modeling pipeline, they were free to choose the basic units of the language model, i.e., its *alphabet*. For example, in his seminal work, Hale (2001) trained a probabilistic context-free grammar over units derived from the Penn Treebank (Marcus, Santorini, and Marcinkiewicz, 1993), i.e., units that follow the Penn Treebank's tokenization scheme.[2] However, as language models grew—both in parameter count and in the size of their

[1]This probability is understood as the comprehender's own predictive distribution over upcoming linguistic units, derived from an unobserved human language model. Empirical studies typically approximate this distribution with language models trained on natural language text.

[2]In line with the fashion of the time, Hale (2001) populated the model's vocabulary with high-frequency words from the training portion of the Penn Treebank, together with a distinguished out-of-vocabulary symbol.

training corpora—it became inconvenient, if not infeasible, to train models from scratch for each study using the proper alphabet.

Within psycholinguistics, the string-mismatch problem (Section 1.1) takes a particular form: with the shift toward large pretrained models (Oh and Schuler, 2023; Wilcox et al., 2023; Wilcox et al., 2020), experimenters are no longer free to attune the model's alphabet to their datasets. Instead, they inherit a fixed tokenization, e.g., byte-pair encoding (Gage, 1994; Sennrich, Haddow, and Birch, 2016), whose units generally do not coincide with linguistically meaningful ones (Church, 2020; Hofmann, Pierrehumbert, and Schütze, 2021; Nair and Resnik, 2023). This mismatch has given rise to a methodological infelicity. Researchers must reconcile the gap between their desired units and the model's alphabet (see Figure 6.1), typically through bespoke post hoc procedures that impose unit boundaries on token strings (Nair and Resnik, 2023; Oh and Schuler, 2024; Pimentel and Meister, 2024; Wilcox et al., 2023; Wilcox et al., 2020). These heuristics vary widely between papers and muddle the interpretation of the units.

Units are not the only level at which we may seek an analysis. For instance, one may wish to relate surprisal to discourse structure (Tsipidi et al., 2025, 2024) by aggregating word-level surprisals into a discourse-level region of interest (ROI; Giulianelli et al., 2024). Beyond the choice of units, then, the modeler must also determine the ROIs, i.e., how units are aggregated into predictors. Units and ROIs often coincide one-to-one, but need not: ROIs may span multiple units or even overlap, unlike units themselves. One might *a priori* construct a language model with discourse-level units, but this requires a more permissive definition than the one given here, because

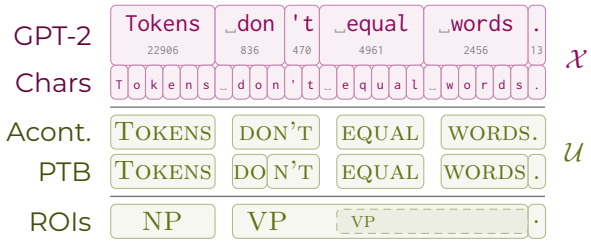


Figure 6.1: The string *Tokens don't equal words.* at three levels: two symbol alphabets, two unit inventories, and regions of interest (ROIs) derived from the sentence's constituency parse: NP, VP (with a nested inner VP shown dashed), and punctuation. The contraction *don't* is split three ways: GPT-2 yields `don`|`'t`, Penn Treebank (PTB) yields DO|N'T, and the acontextual inventory keeps it as one unit DON'T. The period in WORDS. is similarly attached in the acontextual inventory but separate in the contextual one.

discourse structures are nested like syntactic constituents. See Figure 6.1 for an example of nested constituent ROIs.

Our proposal is straightforward: the experimenter first chooses the unit inventory best suited for their analysis. Then, if the language model cannot be retrained, it should be converted to the chosen unit inventory, e.g., by composing the language model with an appropriate function (Chapter 3). On this view, tokenization is little more than an implementation detail that should be of no scientific importance. The unit of analysis is a modeling choice and should be selected to match the scientific question at hand. Depending on the goal, units may be taken to be the model's *token* alphabet (Beinborn and Pinter, 2023; Nair and Resnik, 2023), defined by an explicit segmentation scheme, such as simple delimiter rules (Oh and Schuler, 2024; Pimentel and Meister, 2024), or even derived using contextual segmentation rules, such as the Penn Treebank guidelines (PTB; Marcus, Santorini, and Marcinkiewicz, 1993) or Universal Dependencies (UD; Nivre et al., 2017).

6.1 Units

From the cognitive perspective, linguistic utterances are generally taken to be divisible into discrete units.[3] The question of what constitutes a linguistic unit has been debated at least since Saussure (1997) and Bloomfield (1933), and remains contentious to this day (Haspelmath, 2011; Murphy, 2024). Informally, a unit is a discrete segment of linguistic structure—a phoneme, morpheme, word, or phrase—that serves as an atom of analysis at a chosen level of description. It is generally agreed that linguistic units exist at various levels, i.e., utterances can be divided into clauses, clauses into words, words into morphemes, morphemes into phonemes, and phonemes into phones—even while the notion of a word is a hotly contested one (Dixon and Aikhenvald, 2002; Haspelmath, 2011). Each of these levels provides a valid granularity at which one can decompose an utterance (Hockett, 1958). The choice of level is not merely a technical nuisance but a substantive commitment about the granularity at which cognitive processing is modeled. When—as is common in practice—the model's alphabet is taken to coincide with the unit inventory, this choice directly determines the events to which probability is assigned and, consequently, the quantities that enter the linking hypothesis relating surprisal to behavioral data.

[3]The claim that speech is divisible into discrete units is itself an idealization. The continuous acoustic signal is carved into discrete segments by convention. The mapping from articulatory gestures to perceived phonemes involves substantial coarticulation and context-dependence (Liberman et al., 1967).

Despite this importance, the choice of units has received comparatively little attention in surprisal theory. In practice, most studies inherit their units from the tokenizer of a pretrained language model or from the segmentation conventions of a particular eye-tracking corpus, treating the unit inventory as fixed rather than as a variable to be controlled. This obscures a logically prior question: what *should* the units be? Answering it requires separating $\mathcal{U}$, the countable (but not necessarily finite) unit inventory over which we wish to define surprisal, from $\mathcal{X}$, the alphabet of the language model. We make this separation explicit below. Rather than advocating for a specific unit inventory, we develop a general formalism—in a slogan, "bring your own units"—that is compatible with any choice the modeler wishes to make, irrespective of the language model's native alphabet. To that end, we assume the modeler has a countable unit inventory $\mathcal{U}$. This set may be finite, e.g., the set of phonemes in a language, or countably infinite, e.g., the set of orthographic words.[4]

A linguistic **utterance** is a string of units $\boldsymbol{u} \in \mathcal{U}^*$. A **unit parser** is a stochastic map $\rho\colon \mathcal{X}^* \rightsquigarrow \mathcal{U}^*$ that maps each symbol string $\boldsymbol{x} \in \mathcal{X}^*$ to a probability distribution $\rho(\cdot \mid \boldsymbol{x})$ over unit strings. In many languages, parsing into units is inherently ambiguous. We give a canonical example of such ambiguity from Mandarin Chinese.

Example 6.1 — Segmentation ambiguity in Mandarin

Let $\mathcal{X}$ be the set of Chinese characters, and let $\mathcal{U} = \mathcal{X}^+$. Consider the string $\boldsymbol{x} =$ 乒乓球拍卖完了. The unit parser ρ ought to assign positive probability to (at least) two unit strings:

(1) a. 乒乓球 · 拍卖 · 完了
pīngpāngqiú pāimài wánle
ping pong ball auction finish-PERF
"The ping-pong ball auction is over."

b. 乒乓球拍 · 卖 · 完了
pīngpāngqiúpāi mài wánle
ping-pong-paddle sell finish-PERF
"The ping-pong paddles have been sold."

The character 拍 (*pāi*) is the pivot: it can end the compound 球拍 (*qiúpāi*, 'paddle') or begin 拍卖 (*pāimài*, 'auction'), yielding different unit strings from the same symbol string.

[4] In addition to neologism, recursive morphological processes, e.g., compounding and derivation, produce unboundedly many distinct word forms, yielding a countably infinite set of orthographic words. See Pinker (1994, Ch. 5).

Our choice of a Chinese example is strategic. In many languages that use whitespace in their orthography, e.g., English and most other European languages, parsing into units is far more deterministic. For example, the Penn Treebank tokenization convention (Marcus, Santorini, and Marcinkiewicz, 1993) assigns a unique segmentation to every string. In what follows, we therefore make the simplifying assumption that ρ is *deterministic*: for every $\boldsymbol{x} \in \mathcal{X}^*$, there is exactly one unit string $\boldsymbol{u}$ with $\rho(\boldsymbol{u} \mid \boldsymbol{x}) = 1$. Under this assumption, ρ reduces to a total function $\mathcal{X}^* \to \mathcal{U}^*$. In the remainder of this chapter, we write $\rho(\boldsymbol{x}) = \boldsymbol{u}$ and treat ρ as a function. We call the unit parser's inverse its **realization** $\rho^{-1} \subseteq \mathcal{U}^* \times \mathcal{X}^*$. In general, the realization is a relation, because one unit string may correspond to multiple symbol strings. This is true even in English.

Example 6.2 — Whitespace does not change the units

Let $\mathcal{X}$ be the set of cased ASCII characters, and let $\mathcal{U} = \{\text{HALE}, \text{CITED}, \text{LEVY}\}$. Assuming ρ is deterministic, an English unit parser ought to map the following two symbol strings

(2) a. `Hale␣cited␣Levy.`
b. `Hale␣cited␣␣␣Levy.`

to the same unit string HALE·CITED·LEVY as they differ only in whitespace. Thus, the realization ρ^{-1} is a non-functional relation.

Unit-level distribution. The pushforward of $p_{\mathcal{X}}$ through the total function ρ (Section 3.1) gives the following distribution over unit strings.

$$p_{\mathcal{U}}(\boldsymbol{u}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x} \in \rho^{-1}(\boldsymbol{u})} p_{\mathcal{X}}(\boldsymbol{x}), \tag{6.1}$$

where the realization $\rho^{-1}(\boldsymbol{u}) = \{\boldsymbol{x} \in \mathcal{X}^* \colon \rho(\boldsymbol{x}) = \boldsymbol{u}\}$ is the preimage of $\boldsymbol{u}$ under ρ (Section 3.1). Example 6.2 shows two elements of one such preimage, differing only in whitespace.

6.1.1 Lost in whitespace

Recent work (Oh and Schuler, 2024; Pimentel and Meister, 2024) proposes a formalism for computing the probability of the next unit in context given a symbol-level language model. We identify two conceptual issues with their shared approach—one mathematical and one linguistic.

Unit inconsistency. Both papers formalize the conversion from a symbol-level language model to a unit-level model under three assumptions about

the realization $\rho^{-1} \subseteq \mathcal{U}^* \times \mathcal{X}^*$: (i) ρ^{-1} is a monoid homomorphism,[5] i.e., ρ^{-1} is a function where $\rho^{-1}(u_1 \cdots \cdot u_T) = \rho^{-1}(u_1) \cdots \cdot \rho^{-1}(u_T)$, so each unit is realized in $\mathcal{X}$ independently, and the realization of a sequence of units is the concatenation of the individual realizations; (ii) $\mathcal{X}$ can be partitioned into two disjoint subsets Σ_1 and Σ_2, where Σ_1 contains symbols that mark a unit boundary and Σ_2 does not; (iii) each unit maps to a symbol string in $\Sigma_1 \circ \Sigma_2^*$, i.e., one boundary-marking symbol followed by zero or more continuation symbols. Assumption (i) implies that ρ^{-1} is a function. Assumption (ii) is motivated by the fact that many tokenizers prepend a whitespace character to the first token of each word, so that ␣ signals a unit boundary.[6] Consider the two symbol strings

(3) a. `Hale␣cited␣Levy.`
b. `Levy␣cited␣Hale.`

which we would naturally expect to correspond to the following unit strings

(4) a. $\textsc{Hale}_1 \cdot \textsc{cited} \cdot \textsc{Levy}$
b. $\textsc{Levy} \cdot \textsc{cited} \cdot \textsc{Hale}_2$

where $\textsc{Hale}_1 = \textsc{Hale}_2$. However, assumptions (i)–(iii) make such an equivalence impossible. In the first string, $\textsc{Hale}_1$ is string-initial so $\rho^{-1}(\textsc{Hale}_1) =$ BOS`Hale`, where BOS is the beginning-of-string symbol from Footnote 6. In the second, $\textsc{Hale}_2$ is preceded by ␣ so $\rho^{-1}(\textsc{Hale}_2) =$ ␣`Hale`. Because assumptions (i)–(iii) force ρ^{-1} to be a function, we have $\textsc{Hale}_1 \neq \textsc{Hale}_2$. Consequently, HALE is forced to be two distinct units depending on its context. Thus, the formalism cannot provide a coherent language model over units—the identity of a unit should not depend on whether it begins a string. If we retain assumption (i), changing from leading- to trailing-whitespace attribution moves the exception from the first unit to the last rather than removing it, as Example 6.4 shows. Our framework sidesteps this problem because ρ^{-1} is a relation, not a function: the unit HALE can stand in the realization relation to *both* BOS`Hale` and ␣`Hale`, so a single unit suffices regardless of context.

Linguistic adequacy. A second concern is linguistic. The approach of Oh and Schuler (2024) and Pimentel and Meister (2024) implicitly assumes that units can be recovered by grouping symbols in $\mathcal{X}$ according to the partition

[5] The monoids in question are $(\mathcal{U}^*, \cdot, \varepsilon)$ and $(\mathcal{X}^*, \cdot, \varepsilon)$, i.e., the free monoids over $\mathcal{U}$ and $\mathcal{X}$ respectively, with concatenation as the binary operation and the empty string as the identity.

[6] A notable exception occurs at the beginning of a string: the first symbol belongs to Σ_2 regardless of the unit it represents. One can encode this by introducing an additional copy of all symbols in Σ_1 with a distinguished BOS symbol prepended.

$\Sigma_1 \sqcup \Sigma_2$ from assumption (ii). But byte-pair encoding is a compression algorithm: the boundaries it induces are artifacts of corpus frequency, not of morphological or syntactic structure. Moreover, their framework requires every symbol to be classified as either a boundary marker (in Σ_1) or a continuation symbol (in Σ_2), uniformly across all contexts. Yet a comma is word-internal in `1,000` but marks a clause boundary in `end, he said`. An apostrophe is word-internal in `don't` but possessive-marking in `cat's`. No fixed partition can capture such distinctions in general, since the same character serves different roles in different environments. This concern is far from pedantic. Clark et al. (2025, §2.2.1) exclude words connected to punctuation from their regression analysis because the probability calculation includes attached punctuation in the whitespace-delimited word.

6.1.2 Regular unit inventories

A technical challenge arises when the unit inventory $\mathcal{U}$ is infinite, e.g., the set of all whitespace-delimited words, because language models operate over finite alphabets by definition. The key observation is that even an infinite $\mathcal{U}$ can be finitely represented whenever each unit is itself a string over some finite alphabet, i.e., $\mathcal{U} \subseteq \Xi^*$, and $\mathcal{U}$ forms a *regular* subset of Ξ^*.[7] We call this the **regularity assumption**.

For many abstract linguistic units, the regularity assumption is well-motivated, e.g., it is widely established that phonotactic constraints are regular (Heinz, 2018; Kaplan and Kay, 1994), as are many morphotactic rules (Beesley and Karttunen, 2003; Koskenniemi, 1983).[8] For unit inventories defined by such constraints, regularity of $\mathcal{U}$ is a mild assumption in practice. Under this assumption, we can reduce operations on a potentially infinite $\mathcal{U}$ to operations over regular sets as follows. Let $\textsc{sep} \notin \Xi$ be a distinguished separator symbol, and write $\boldsymbol{\xi}_u \in \Xi^*$ for the string a unit u spells. We define

$$\begin{aligned} h \colon \mathcal{U} &\to \Xi^* \textsc{sep} \\ u &\mapsto \boldsymbol{\xi}_u \textsc{sep}, \end{aligned} \tag{6.2}$$

which appends $\textsc{sep}$ to each unit's underlying string.[9] This extends to a monoid homomorphism $\mathcal{U}^* \to (\Xi^* \textsc{sep})^*$ by defining $h(u_1 \cdot \cdots \cdot u_T) \stackrel{\text{def}}{=} h(u_1) \cdot \cdots \cdot h(u_T)$.

[7] A language is *regular* if it is accepted by a finite automaton. See Pin (2025) for a comprehensive treatment.

[8] A notable exception is productive total reduplication, which requires unbounded copying and therefore falls outside the one-way finite-state framework used here. However, Dolatian and Heinz (2018) argue that 2-way finite-state transducers can nonetheless model it.

[9] It will be important that h be **prefix-free**, i.e., there do not exist distinct units $u_1, u_2 \in \mathcal{U}$ such that $h(u_1) \preceq h(u_2)$.

Note that $h(\mathcal{U})$ is regular since $\mathcal{U}$ is regular by assumption and regular sets are closed under concatenation, and so $h(\mathcal{U}^*) = h(\mathcal{U})^*$ is regular by standard closure properties.

6.1.3 A finite-alphabet representation of the unit parser

We now introduce a *computational* formalism for describing $\rho\colon \mathcal{X}^* \to \mathcal{U}^*$. First, let $\mathcal{Y} \stackrel{\text{def}}{=} \Xi \sqcup \{\text{SEP}\}$, following Equation (6.2). Then, note that the map $\boldsymbol{x} \mapsto h(\rho(\boldsymbol{x}))$ takes $\mathcal{X}$-strings and maps them to strings over the finite alphabet $\Xi \sqcup \{\text{SEP}\}$. If we can compute this map, we can apply h^{-1} to the output to map back to a unit string. This lets us avoid working directly with the potentially infinite unit inventory $\mathcal{U}$.

For the finite-state definitions and standard operations used here, including composition and input projection, see Section 2.4.

Define $f\colon \mathcal{X}^* \to \mathcal{Y}^*$ by $f(\boldsymbol{x}) \stackrel{\text{def}}{=} h(\rho(\boldsymbol{x}))$. Note that f maps strings between two *finite* alphabets, $\mathcal{X}$ and $\mathcal{Y}$, even when the unit inventory $\mathcal{U}$ is infinite[10] because $\mathcal{U} \subseteq \Xi^*$. If, in addition, f is rational,[11] Chapter 4 provides practical algorithms for evaluating prefix probabilities under the pushforward distribution defined in Equation (3.1). We now extract unit-level probabilities from it. Writing $\overrightarrow{p_{\mathcal{U}}}$ for the prefix probability of $p_{\mathcal{U}}$, the conditional next-unit probability is given by

$$\overrightarrow{p_{\mathcal{U}}}(u \mid \boldsymbol{u}) = \overrightarrow{p_{\mathcal{Y}}}(h(u) \mid h(\boldsymbol{u})). \tag{6.3}$$

The simplicity of Equation (6.3) follows from the injectivity of h, which holds by construction, and from its prefix-freeness (Footnote 9).

Example 6.3 — Computing a next-unit probability with a trailing SEP

Let the unit inventory be $\mathcal{U} = \{\text{A}, \text{AB}\}$ over $\Xi = \{\text{a}, \text{b}\}$, with the parser mapping each whole source string to a single unit and the source model splitting its mass evenly:

$$\rho(\text{a}) = \text{A}, \qquad \rho(\text{ab}) = \text{AB}, \qquad p_{\mathcal{X}}(\text{a}) = p_{\mathcal{X}}(\text{ab}) = \tfrac{1}{2}.$$

The unit-level model should therefore give $\overrightarrow{p_{\mathcal{U}}}(\text{A}) = \frac{1}{2}$.

[10] There is a relevant line of work that extends classical finite automata and transducers with transitions labeled by predicates over potentially infinite alphabets, known as *symbolic* finite automata and transducers (D'Antoni and Veanes, 2017; Noord and Gerdemann, 2001; Veanes et al., 2012).

[11] The regularity assumption is necessary for rationality of f whenever ρ surjects onto $\mathcal{U}^*$.

Compute it using Equation (6.3). The transducer f maps

$$\mathsf{a} \mapsto \mathsf{a}\textsc{sep}, \qquad \mathsf{ab} \mapsto \mathsf{ab}\textsc{sep},$$

and $\overrightarrow{p_{\mathcal{Y}}}(h(\textsc{a}))$ sums the source strings whose image extends $\mathsf{a}\textsc{sep}$. Only a qualifies (the image of ab continues a with b rather than $\textsc{sep}$), so

$$\overrightarrow{p_{\mathcal{Y}}}(\mathsf{a}\textsc{sep}) = p_{\mathcal{X}}(\mathsf{a}) = \tfrac{1}{2}.$$

The trailing $\textsc{sep}$ marks the unit's right edge.

Under the mirror convention h_{l} ($\textsc{sep}$ marking onsets), the images become $\textsc{sep}\mathsf{a}$ and $\textsc{sep}\mathsf{ab}$, and *both* extend $h_{\mathrm{l}}(\textsc{a}) = \textsc{sep}\mathsf{a}$:

$$\overrightarrow{p_{\mathcal{Y}}}(\textsc{sep}\mathsf{a}) = p_{\mathcal{X}}(\mathsf{a}) + p_{\mathcal{X}}(\mathsf{ab}) = 1 \neq \tfrac{1}{2},$$

the mass of $\textsc{ab}$ having leaked into $\textsc{a}$. Section C.1 gives the general argument.

Figure 6.2 shows the direct and inverse relationships among symbol strings, separator-annotated strings, and unit strings.

6.2 Regions of interest

Having established a finite-alphabet representation of the unit parser, we now turn to the aggregation of unit-level surprisal over regions of interest. Experimenters often study predictions at spans that cover multiple word-level units, such as sentences (Giulianelli, Wallbridge, and Fernández, 2023; Lau, Clark, and Lappin, 2017; Meister et al., 2021b), dialogue turns (Wallbridge, Bell, and Lai, 2023; Wallbridge, Lai, and Bell, 2022), or discourse segments (Tsipidi et al., 2025, 2024). To describe such spans formally, let $\mathcal{U}^+ = \mathcal{U}^* \setminus \{\varepsilon\}$.

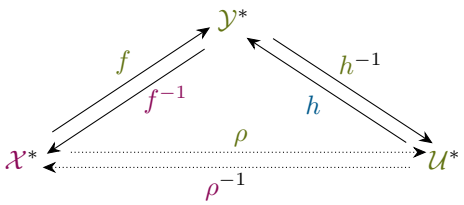


Figure 6.2: The unit parser $\rho\colon \mathcal{X}^* \to \mathcal{U}^*$ passes through $\mathcal{Y}^*$: the transducer f maps symbol strings to $\textsc{sep}$-annotated strings, and h^{-1} splits on $\textsc{sep}$ and maps each segment to the unit it spells, recovering the unit string.

Given an utterance $\boldsymbol{u} = u_1 \cdots u_T \in \mathcal{U}^+$ of T units, we write $\boldsymbol{u}_{[i,j)} = u_i \cdots u_{j-1}$ for $1 \leq i < j \leq T+1$ and refer to it as a **region of interest** (ROI; Giulianelli et al., 2024).[12]

To predict reading time for an ROI $\boldsymbol{u}_{[i,j)}$ from a language model $p_{\mathcal{U}}$ over $\boldsymbol{u} \in \mathcal{U}^*$, one must specify how to combine unit-level surprisals. The standard approach is to sum surprisals over the ROI (Nair and Resnik, 2023; Smith and Levy, 2013). Importantly, this sum yields the surprisal of the character sequence but omits the probability of SEP or EOS signaling the unit boundary, so it is not directly comparable with unit-level surprisal.

Computing ROI-level surprisal requires that ROI boundaries be compatible with finer-grained unit boundaries. Otherwise, a unit may overlap two ROIs and cannot be unambiguously assigned to either. For example, consider the stimulus *Predictive power* with characters as units. A psycholinguist studying parafoveal preview (Blanchard, Pollatsek, and Rayner, 1989; Rayner, 1975; Rayner et al., 1982) might define each ROI as the first three characters of a word. However, as shown in Ex. (5), the ROI P·R·E spans parts of two GPT-2 (Radford et al., 2019) tokens (`P` and `redict`), so the model's token-level probabilities alone cannot yield the surprisal of this character span. To resolve such mismatches, we can transform a language model from its native token alphabet to the character alphabet (Vieira et al., 2025a).

(5) a. P·R·E·D·I·C·T·I·V·E·␣·P·O·W·E·R
Character units (ROIs underlined)

b. `P redict ive ␣power`
47 17407 425 1176 GPT-2 tokens

6.3 From surprisal to reading times

Surprisal theory posits that the incremental processing difficulty of language comprehension is a function of how unexpected an upcoming linguistic unit is given its context. The theory assumes an implicit human language model p_{H}, and predicts that the processing effort incurred by a unit is monotonically related to its surprisal. In practice, empirical tests of surprisal theory rely on large pretrained language models as proxies for p_{H} (Kuribayashi, Oseki, and Baldwin, 2024; Oh and Schuler, 2023; Shain et al., 2024; Wilcox et al., 2023). Additionally, evaluating the theory requires a **linking hypothesis**, i.e., a specification of how surprisal maps onto an observable dependent variable such as reading time. Much attention has been devoted to the functional form of this mapping, whether processing effort scales linearly (Shain et al.,

[12] Also referred to as an area of interest (Schotter and Dillon, 2025).

2024; Smith and Levy, 2013; Wilcox et al., 2023), sublinearly (Brothers and Kuperberg, 2021), or superlinearly (Hoover et al., 2023) with surprisal, while Xu et al. (2023) find that the shape depends on the language model. However, as we argue in this chapter, the choice of units and the aggregation strategy are also consequential.

The training set $\{\boldsymbol{u}^n\}_{n=1}^N$ consists of N distinct utterances. We write T_n for the number of units in the n^{th} utterance. For each u_t^n given preceding context $\boldsymbol{u}_{<t}^n$, we measure a reading time $r_\pi(u_t^n, \boldsymbol{u}_{<t}^n)$ from one of the P participants, indexed by π. Because fixation durations are strictly positive and right-skewed, we model reading times with a log-normal generalized additive mixed model (GAMM; Wood, 2017). At position $t \in [T_n]$ of utterance n, let $\mathbf{x}_t^n = (x_{1,t}^n, \dots, x_{J,t}^n)^\top$ denote the vector of J predictors. We model

$$\log r_\pi(u_t^n, \boldsymbol{u}_{<t}^n) = \mu_\pi(u_t^n, \boldsymbol{u}_{<t}^n) + \epsilon, \tag{6.4}$$

where $\epsilon \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise, σ^2 is the residual variance, and the mean on the log-reading-time scale is given by

$$\mu_\pi(u_t^n, \boldsymbol{u}_{<t}^n) = \sum_{j=1}^{J} f_j(x_{j,t}^n) + z_\pi(\mathbf{x}_t^n). \tag{6.5}$$

Each f_j is a penalized smooth function of the j^{th} component $x_{j,t}^n$ of $\mathbf{x}_t^n$, so that the relationship between each predictor and reading time is learned nonparametrically. See Section 6.3.1 for a discussion of the predictors. The term $z_\pi(\mathbf{x}_t^n)$ captures participant-level random effects: a random intercept and by-participant random slopes for each predictor. See Section C.5 for the full specification.

6.3.1 Baseline predictors

A unit's **length** and **frequency** are standard baseline controls in eye-tracking regressions (Demberg and Keller, 2008; Goodkind and Bicknell, 2018; Smith and Levy, 2013): short, high-frequency units are more likely to be skipped and, when fixated, receive shorter fixation durations than longer, lower-frequency ones (Kliegl et al., 2004; Rayner and Raney, 1996). Accordingly, evaluations of contextual surprisal typically include both as baseline predictors (Kuribayashi, Oseki, and Baldwin, 2024; Opedal et al., 2024; Wilcox et al., 2023), with frequency operationalized as **unigram surprisal**.[13] Prior work either computes word frequencies on held-out data (Pimentel and Meister, 2024)

[13] See Shain (2019, 2024) and Opedal et al. (2024) for discussion of unigram surprisal in analyzing reading times.

or relies on precompiled lexical resources (e.g., Opedal et al., 2024; Re et al., 2025; Wilcox et al., 2023), using toolkits such as Speer (2022). However, these convenient methods introduce two nontrivial mismatches. First, Speer (2022) conflates distinct orthographic forms by stripping punctuation. For instance, it assigns AND the same probability as AND·,. Second, the resulting unigram distribution is not aligned with the language model used to derive contextual surprisal, complicating comparisons between frequency and contextual predictability. We thus follow Hopton et al. (2026) and estimate unigram surprisal directly from the language model: we sample text from the LM, process each sample through the transduced LM to obtain per-unit conditional probabilities, and average these over all positions. The resulting unigram distribution is consistent with the model's own distribution and is naturally defined for every unit in our inventories. See Section C.4 for additional details. We include unigram surprisal estimated in this way as a baseline predictor in all analyses. Finally, to account for spillover effects, we include the predictors for the two preceding units as controls (Rayner, Carlson, and Frazier, 1983).

6.3.2 Predictive power

We evaluate the contribution of surprisal by comparing two instances of the model in Equations (6.4) and (6.5): a **baseline model** $\widetilde{\varphi}$, in which the log-mean $\widetilde{\mu}_\pi(u_t^n, \boldsymbol{u}_{<t}^n)$ depends only on control predictors (unit length, unigram surprisal, and their spillover lags, Section 6.3.1), and a **target model** φ that adds contextual surprisal and its spillover lags (see Section C.5 for the full specification). We fit both models on the N training utterances and evaluate them on a held-out test set of M utterances, where utterance m spans T_m positions. Let $\mathcal{I}$ denote the collection of held-out (utterance, position) indices (m, t), one for each observed participant reading time, where $m \in [M]$ and $1 \leq t \leq T_m$. For brevity, we suppress the participant index associated with each entry and write $r_t^m \stackrel{\text{def}}{=} r_\pi(u_t^m, \boldsymbol{u}_{<t}^m)$ and $\mu_t^m \stackrel{\text{def}}{=} \mu_\pi(u_t^m, \boldsymbol{u}_{<t}^m)$ (and $\widetilde{\mu}_t^m$ for the baseline). Writing the log-normal density from Equation (6.4) as

$$\varphi\big(r_t^m \mid \mu_t^m, \sigma^2\big) \stackrel{\text{def}}{=} \frac{1}{r_t^m\, \sigma\sqrt{2\pi}} \exp\left(-\frac{(\log r_t^m - \mu_t^m)^2}{2\sigma^2}\right), \tag{6.6}$$

we measure the mean per-observation improvement in held-out log-likelihood, which is defined as

$$\Delta_{\text{llh}} \stackrel{\text{def}}{=} \frac{1}{|\mathcal{I}|} \sum_{(m,t)\in\mathcal{I}} \log \frac{\varphi(r_t^m \mid \mu_t^m,\, \sigma^2)}{\widetilde{\varphi}(r_t^m \mid \widetilde{\mu}_t^m,\, \widetilde{\sigma}^2)}, \tag{6.7}$$

where μ_t^m and $\widetilde{\mu}_t^m$ are the predicted log-means from the target and baseline models at position t of held-out utterance m, and σ and $\widetilde{\sigma}$ are the corresponding residual standard deviations, both estimated on the training set. A positive Δ_{llh} indicates that contextual surprisal captures variance in reading times beyond the baseline controls.

6.4 Experiments

We now evaluate the predictive power of surprisal theory under four unit inventories.

6.4.1 Unit inventories

Tokens. The simplest option is to use the model's native tokens as units, i.e., $\mathcal{U} = \mathcal{X}$. This is a natural choice when the objective is to characterize the model itself, e.g., when comparing the impact of token granularity on surprisal (Oh and Schuler, 2025), or to evaluate the cognitive plausibility of token-like representations (Beinborn and Pinter, 2023; Nair and Resnik, 2023).[14] In other experimental paradigms, however, model tokens are a poor fit, as they rarely align with linguistically meaningful units (Church, 2020; Hofmann, Pierrehumbert, and Schütze, 2021; Nair and Resnik, 2023). Another limitation of tokens is their coarse, model-dependent granularity, which can obscure effects that are naturally defined at finer spatial scales in the stimulus (Rayner, 1975; Schotter, Angele, and Rayner, 2012). See, for example, ex. (5).

Characters. At the other extreme, individual characters can constitute units. Character-level surprisal may be useful when the ROIs are sublexical, such as punctuation (Hill and Murray, 2000; Hirotani, Frazier, and Rayner, 2006; Rayner, Kambe, and Duffy, 2000), morphological structure (Nair and Resnik, 2023), the first few characters to predict the skip rate of a word (Blanchard, Pollatsek, and Rayner, 1989; Rayner et al., 1982), or as sublexical information used to improve surprisal estimates for larger units (Oh, Clark, and Schuler, 2021). We study character-level units in their own right, not merely as building blocks for computing ROI-level predictors.

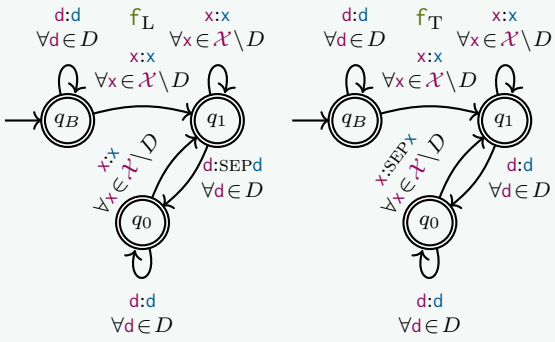
Acontextual words. A common choice is to define units using explicit orthographic rules. For instance, one could define a word-like notion by

[14] BPE can be encoded as a finite-state machine (Berglund, Martens, and Merwe, 2024; Berglund and Merwe, 2023).

choosing the delimiter set Σ_1 from the partition $\Sigma_1 \sqcup \Sigma_2$ of $\mathcal{X}$ introduced in Section 6.1.1, such as whitespace (Pimentel and Meister, 2024; Wilcox et al., 2023), and splitting on those delimiters (a design choice here is whether to delete the delimiters or leave them in the resulting units). As illustrated in Example 6.4, this kind of segmentation can be implemented with a finite transducer. However, this is a modeling convenience inherited from how popular eye-tracking corpora such as Dundee (Kennedy, Hill, and Pynte, 2003), Provo (Luke and Christianson, 2018), or MECO (Siegelman et al., 2022) distribute their data, and is not meant to represent a linguistically adequate notion of a word. Its simplicity is also its main limitation: splitting on delimiters cannot express context-dependent boundaries. Several recent studies have excluded all words attached to punctuation (Clark et al., 2025; Gruteke Klein et al., 2024; Nair and Resnik, 2023), even though research has long reported systematic effects of punctuation on reading behavior (e.g., pauses and longer first-pass times around commas) (Hill and Murray, 2000; Hirotani, Frazier, and Rayner, 2006; Rayner, Kambe, and Duffy, 2000).

Example 6.4 — Limits of delimiter segmentation

The two separator-insertion transducers below implement the two acontextual segmentation conventions. At each internal boundary, $\mathtt{f}_{\mathrm{L}}$ (left) inserts SEP before the delimiter sequence and $\mathtt{f}_{\mathrm{T}}$ (right) after it. With whitespace as the delimiter, these distinguish leading- and trailing-whitespace decoding (Oh and Schuler, 2024; Pimentel and Meister, 2024). See Section C.2.2 for discussion.



Applying both to the string of Example 6.2 with $D = \{␣\}$ gives the following segmentations.

$\mathtt{f}_{\mathrm{L}}$ (leading)	HALE · ␣CITED · ␣LEVY.
$\mathtt{f}_{\mathrm{T}}$ (trailing)	HALE␣ · CITED␣ · LEVY.

The two conventions differ only in which unit contains the whitespace. Each leaves one edge unit without the whitespace carried by the others. Under leading attribution, this is the initial HALE. Under trailing attribution, it is the final LEVY. Replacing the string by `Levy cited Hale.` therefore changes the unit containing `Hale` under either convention. Switching from leading to trailing attribution moves the edge exception rather than removing it (Section 6.1.1).

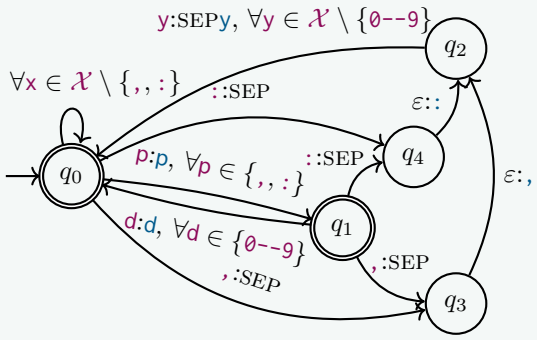
The final unit exposes another limitation. Under *both* conventions, the period remains attached to `Levy`, so the inventory contains LEVY. rather than LEVY. This does not match an ROI that treats the word and punctuation separately. A context-independent delimiter set cannot separate sentence-final periods without also splitting periods inside expressions such as `3.14` and `Ph.D.`.

Contextual words. The acontextual assumption, i.e., that a fixed partition of $\mathcal{X}$ suffices to determine unit boundaries, is linguistically inadequate, because the same symbol can mark a boundary in one context but not another. Thus, instead of defining units via an explicit set of delimiters, it is common practice to use contextual segmentation rules, such as the Penn Treebank guidelines (Marcus, Santorini, and Marcinkiewicz, 1993), or Universal Dependencies (Nivre et al., 2017). Such units are linguistically informed, but introduce an additional challenge with token-level language models, where tokens are not compatible with the resulting units. Here we follow the construction in Section A.2.5 by encoding each rule[15] as a finite transducer and then compose them left to right to obtain $\mathtt{f}_{\text{ptb}}$. See Section C.2.3 for additional details. In contrast to the acontextual segmentation in Example 6.4, this transducer determines word boundaries using contextual information. Consider, for example, the rule worked through in Example 6.5, which inserts a separator SEP before a comma (`,`) or a colon (`:`) only if the following symbol is not a digit.

Example 6.5 — Segmenting commas and colons by context

The rule below generalizes Figure 4.2. It splits a comma or colon off as its own unit, surrounded by SEPs, only when the *following* symbol is not a digit.

[15] For an overview of the individual tokenization rules, see `TreebankWordTokenizer`.



Apply the rule to `key:value`. From q_0, there are two transitions with the colon as input. Only the path through q_4 can be extended to an accepting path for this string. Its continuation from q_2 admits `v` because `v` is not a digit. The output labels along this path place separators around the colon, yielding KEY · : · VALUE. The alternative through q_1 has no continuation with `v` as input.

For `12:30`, only the path through q_1 can be extended to an accepting path because q_1 has a transition with `3` as input, whereas q_2 does not. The output labels along this path copy the colon unchanged, leaving 12:30 as one unit. Thus the following symbol determines which colon transition can occur on an accepting path.

6.4.2 (Re-)processing the MECO corpus

To obtain fixation data for each unit inventory discussed in Section 6.4.1, we process the raw fixation data from the MECO dataset (Siegelman et al., 2022), which contains scanpaths from 46 readers recorded while reading 12 short excerpts drawn from Wikipedia articles. We use the English portion of the dataset. Following Re et al. (2025), we first obtain the unprocessed fixation data and use the predefined bounding boxes to match fixations with individual characters. We then tokenize the raw text and aggregate the fixation durations within the boundaries of each unit, to obtain three commonly used (Rayner, 1998) reading time measurements $r(u_t, \boldsymbol{u}_{<t})$: **first-fixation duration**, the duration of the first fixation on unit u_t; **gaze duration**, the sum of all first-pass fixations on u_t; and **total reading time**, the sum of all fixations on u_t, including any refixations. Fixations on whitespace characters are credited to whichever unit the transducer assigns that whitespace to. We exclude unfixated units and retain per-reader observations for use with mixed-effects models (Section C.5). For the character inventory, we also exclude observations whose surprisal is exactly zero, which arise at sub-token

byte positions. In Section C.3.2, we visualize the resulting units and fixations, and Table C.2 reports the observation counts at each step of the pipeline.

6.4.3 Estimating surprisal

We use GPT-2 Small (Radford et al., 2019) as our symbol language model $p_{\mathcal{X}}$, following previous work (Oh and Schuler, 2023). For the token inventory, surprisal is read directly from $p_{\mathcal{X}}$. For the character inventory, we obtain a byte-level model using the method of Vieira et al. (2025a), while for the word inventories we compose $p_{\mathcal{X}}$ with the appropriate finite transducer. Section C.4 details the scoring pipeline for contextual and unigram surprisal and other hyperparameters (Table C.5).

6.4.4 Analysis

We fit the log-mean of Equation (6.4) for both baseline and target models as the GAMM of Equation (6.5), which estimates the contribution of each predictor through a penalized smooth function. The residual standard deviations (σ and $\widetilde{\sigma}$) are estimated on the log scale of the training set. To assess generalization, we perform leave-one-out cross-validation by trial (12 folds): in each fold, we fit both models on 11 trials and compute Δ_{llh} (Equation (6.7)) on the held-out trial. We report the per-observation Δ_{llh} with 95% confidence intervals obtained by trial-level bootstrap (1000 iterations). Significance of Δ_{llh} is assessed by a one-sided paired sign-flip permutation test over held-out log-likelihoods. See Section C.5 for details.

6.4.5 Results

Figure 6.3 shows our results across unit inventories and reading-time measures. Detailed results are given in Table C.8. Adding surprisal as a predictor yields significant improvements in Δ_{llh} over the baseline for word-like inventories on later reading-time measures (gaze duration and total reading time, all $p < 0.001$ by paired permutation test), in line with previous findings (Goodkind and Bicknell, 2018; Wilcox et al., 2023). For first-fixation duration, results are significant for tokens, contextual words, and acontextual words under trailing-whitespace attribution, but not under leading-whitespace attribution. The character inventory shows no significant Δ_{llh} on any measure.

A central lesson is that changing the unit of analysis changes the regression problem itself: different inventories induce different observations and controls (length, spillover, unigram surprisal), so absolute log-likelihoods and Δ_{llh} values are not directly comparable across inventories (Table C.2). This lesson

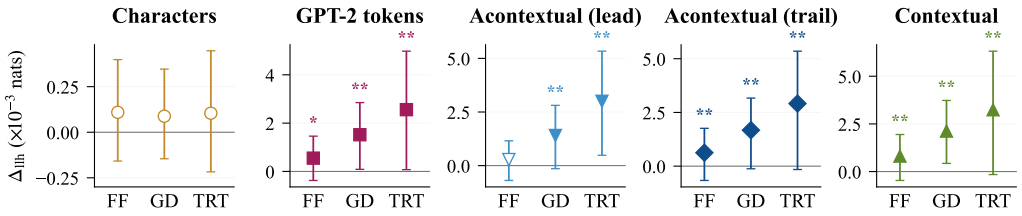


Figure 6.3: Per-observation Δ_{llh} ($\times 10^{-3}$ nats) for each unit inventory across reading-time measures (FF: first fixation, GD: gaze duration, TRT: total reading time). Points and whiskers show the mean and 95% trial-level bootstrap CI from leave-one-out cross-validation by trial. Significance is assessed via a paired permutation test ($^{*}p < 0.05$; $^{**}p < 0.01$). Filled markers denote significant effects. Note that y-axis scales differ across panels: log-likelihoods are not comparable across inventories because the number and granularity of observations differ.

matters beyond the inventories considered here: existing work has evaluated surprisal at granularities from morphemes (Nair and Resnik, 2023), phonemes (Brodbeck et al., 2022; Tezcan, Weissbart, and Martin, 2023), and syllables (Sohoglu, Beckers, and Davis, 2024) to sentences (Giulianelli, Wallbridge, and Fernández, 2023; Lau, Clark, and Lappin, 2017) and discourse units (Tsipidi et al., 2025, 2024). The framework applies wherever the unit of analysis must be specified, whether the dependent variable is reading time, neural signals (Frank et al., 2013, 2015; Kuribayashi et al., 2025), or a modified predictive distribution such as lossy-context surprisal (Futrell, Gibson, and Levy, 2020) or syntactic surprisal (Arehalli, Dillon, and Linzen, 2022; Demberg and Keller, 2008).

6.5 Reanalysis with unbiased prefix-probability estimates

This section contains material adapted from Snæbjarnarson et al. (2026b).

In the preceding sections, we predict human reading times on the MECO eye-tracking corpus (Siegelman et al., 2022) from next-unit *contextual surprisal* computed with GPT-2 Small under several unit inventories. The PTB-word surprisals come from the per-position beam-summing estimate of Chapter 4, computed with $\texttt{prune}_\tau$. Each next-unit probability is estimated by the ratio $\hat{Z}_t / \hat{Z}_{t-1}$. Pruning lowers both estimates, but it can remove different fractions

of their mass, so the ratio and resulting surprisal can err in either direction.

Beam summing with `prune`$_\tau$ gives no way to determine how the resulting bias affects the PTB-word conclusions. We therefore replace `prune`$_\tau$ with unbiased SWOR pruning, recompute the PTB-word contextual surprisals, and rerun the corresponding fits. The resulting estimate of corpus surprisal is 106 nats lower than the original. We evaluate first-fixation duration, gaze duration, and total reading time. For each measure, Δ_{llh} is the per-observation improvement in held-out log likelihood from adding contextual surprisal to the shared baseline.

For each reading-time measure, we compare fits using surprisal computed by `beam_summing` with either `prune`$_\tau$ or `swor_adaptive`. The model specification, evaluation protocol, and beam-based unigram-surprisal control of Kiegeland et al. (2026) are unchanged (Table 6.1; details in Section C.6). We find no evidence of a predictive difference between the two fits on any reading-time measure (one-sided $p \geq 0.068$ in both directions). Under either estimator, adding contextual PTB-word surprisal improves held-out prediction for all three reading-time measures (Table 6.1). Thus, replacing `prune`$_\tau$ with `swor_adaptive` leaves the PTB-word analysis unchanged.

6.6 Chapter limitations

Unit inventories and ROIs. Our empirical evaluation is restricted to tokens, characters, and two families of word-like segmentations (contextual and acontextual). Beyond these units, researchers have studied several other unit inventories and ROIs, such as discourse units (e.g., clauses or elementary discourse units).

Language and model coverage. Our empirical results are restricted to English: what counts as a word varies with orthography and linguistic traditions, and many languages require segmentation rules different from

Table 6.1: Per-observation improvement in held-out log likelihood from adding PTB-word contextual surprisal on MECO. Values are Δ_{llh} in units of 10^{-3} nats. For `swor_adaptive`, we use ρ=0.1, nominal M=8000, and 10 seeds. $^{*}\,p < 0.05$; $^{**}\,p < 0.01$; $^{***}\,p < 0.001$.

Pruning function	First fixation	Gaze duration	Total reading time
`swor_adaptive`	0.76^{**}	2.05^{***}	3.26^{***}
`prune`$_\tau$	0.81^{**}	2.13^{***}	3.24^{***}

those standard in English (Nivre et al., 2017). In addition, our analysis evaluates only GPT-2 Small on the MECO dataset, using GAMMs to predict reading times. Future studies could therefore broaden the empirical analyses to evaluate unit inventories and ROI choices across models and datasets.

Data requirements. Our analysis requires access to raw fixation data. Many published reading-time corpora distribute only pre-aggregated word-level reading times, and self-paced reading datasets are inherently bound to a fixed segmentation. In such cases, fixations cannot be re-aggregated to alternative unit boundaries, and our approach applies only if the chosen unit inventory is compatible with the corpus's pre-existing segmentation.

Computational costs. Computing surprisal under the transduced language model requires summing source-model mass over the precover of each target prefix, which incurs computational overhead that depends on the transducer and the source language model. In our experiments, we use the pruning-based approximations of Chapter 4, together with the beam-search byte-level models of Vieira et al. (2025a). The resulting throughput (Table C.6) makes contextual surprisal estimation computationally feasible at the scale of the MECO corpus studied here. Estimating unigram surprisal is considerably more demanding, so we parallelize it across samples. See Section C.4 for details. A lighter-weight alternative is to estimate unigram probabilities by transducing the sampled text and directly counting unit occurrences, which sidesteps sequential per-boundary scoring of every candidate unit but can underestimate units that rarely appear in the samples. See Hopton et al. (2026). Another alternative would be a hybrid approach that uses sample counts for frequent units and falls back to the conditional estimate for rare units.

Expressivity. Our framework inherits the expressivity limits of finite-state machinery: we assume the unit parser ρ is deterministic and rational, i.e., realizable by a finite transducer. Phenomena beyond this scope, such as non-rational transformations, e.g., those requiring context-free structure, are left to future work.

Chapter 7

Summary and discussion

This thesis has proposed *transduced language models* (TLMs): the composition of a pretrained language model with a string-to-string transformation encoded by a finite-state transducer. The construction was motivated in Chapter 1 by the *string-mismatch problem*, the gap between the distribution a pretrained model defines and the one a downstream application needs. Common methods for building vocabularies are chosen because the models trained on them perform well. Nothing about that choice reflects an interest in the vocabulary items themselves. They are an implementation detail, and the specific tokenization is typically hidden from users of commercial language model interfaces, who see only the model's decoded output. When an application uses the model's probabilistic interface, however, the units over which that interface is defined start to matter.

7.1 Overview

Chapter 3 formalized the construction of a transduced language model as the pushforward of the source distribution under the function encoded by the transducer. Its prefix probabilities sum source-string probabilities over a *precover* of the target prefix, which decomposes into a prefix-free *quotient* and a *remainder*. The quotient collapses whole cylinders of source strings into single prefix-probability calls, while the remainder contributes exact source-string probabilities. The inference algorithms of Chapters 4 and 5 evaluate or estimate this sum. The construction lifts Vieira et al.'s (2025) strict-prefix-monotone setting to arbitrary functional FSTs, and the finiteness conditions of Section 3.3 identify a class of transducers for which the decomposition can be computed exhaustively.

Chapter 4 showed how to compute the autoregressive interface. The exact algorithm operates directly on the transducer through the lazily determinized

frontier and the IP-universality shortcut, and reduces to the Vieira et al. algorithm on its strict-prefix-monotone special case for subword-to-byte conversion, where every state is IP-universal and every non-ε-input arc emits output. Exhaustive enumeration is exact but can be out of reach, so the chapter's practical mode is probability-mass pruning over the frontier, at the throughputs reported in Section 4.4. Even with this pruning, source-prefix enumeration remains too expensive in the high-entropy DNA setting at realistic sequence lengths (Section 5.2.3).

Chapter 5 replaced biased deterministic pruning with stochastic estimation of the prefix probability. The chapter framed the prefix probability as a normalizing constant and built a sequential Monte Carlo estimator around the quotient–remainder decomposition by combining source-prefix enumeration restricted to live extensions, Rao–Blackwellization of the stop decision, and without-replacement pruning with Horvitz–Thompson reweighting. The estimator meets the three criteria stated at the opening of Section 5.1. It is unbiased (i), its variance decreases empirically as the particle count grows (ii), and it has configurable cost, with adaptive-budget pruning ensuring almost-sure halting even when the decomposition is infinite (iii). At a fixed maximum particle count in the DNA-to-amino-acid experiment, running time grows approximately linearly over tested target lengths even as the compatible source-prefix set grows exponentially.

Chapter 6 applied TLMs to psycholinguistics, where the choice of units has been contested. By converting a single pretrained LM onto new alphabets, the chapter separated two methodological choices that prior work had conflated, the *unit* of analysis and the *region of interest* over which surprisal is aggregated. The chapter's formalism makes clear that defining next-unit contextual surprisal requires a choice of units. Because that choice affects both the regression and its baseline predictors, it should be made explicitly rather than inherited from the language model's tokenizer. Repeating the PTB-word analysis with the unbiased estimator of Chapter 5 left its conclusions unchanged (Section 6.5).

7.2 Research questions

Taken together, the results of this thesis answer the three research questions posed in Section 1.3. A functional finite-state transducer transforms a pretrained model by the pushforward, and the quotient–remainder decomposition expresses the transformed model's prefix probabilities in terms of source-model probabilities (RQ1). The lazily determinized frontier computes the autoregressive interface exactly when the required decompositions are

finite and uses probability-mass pruning for an approximation when exhaustive enumeration is impractical (RQ2). When exact computation is out of reach, the without-replacement estimator gives unbiased prefix-probability estimates (RQ3).

7.3 Limitations

Finiteness. In Lemma 3.1, we give sufficient conditions under which a function encoded by a transducer has a finite quotient–remainder decomposition for every target string. We do not, however, show that these conditions are necessary. Further theoretical work is needed to determine whether they are necessary and, more generally, to characterize exactly when finite decompositions exist.

Empirical scope. The framework is not tied to a particular language-model architecture, but its algorithms require access to the source model's next-symbol probabilities and a transformation realized by a functional finite-state transducer. The experiments are restricted to a handful of source models (GPT-2 and bigram models trained on text or DNA, Llama 3, and Phi-4) and a small set of unit choices (BPE tokens, bytes, characters, whitespace-delimited words, Penn Treebank words, and amino acids). Future evaluations could cover a wider range of models, training corpora, and target units.

Expressiveness. The analysis in the thesis concerns functions encoded by finite-state transducers over finite alphabets. Generalizing to stochastic non-functional transductions and more expressive transformations would require corresponding generalizations of the precover and the universality conditions (see Section 7.4).

Variance. The experiments in Section 5.2.4 show that accumulating quotient and remainder contributions reduces variance, whereas the without-replacement estimators have the lowest variance in the settings studied. Sampling without replacement has established variance-reduction results (Fearnhead and Clifford, 2003), but we do not prove a variance ordering among the complete estimators.

Speed. Scoring a given target under a transduced language model with beam summing and threshold pruning currently runs at around ten target symbols per second for the PTB transducer in the reported setup (Table C.6), and

the full next-symbol distribution is slower still at low thresholds (Table A.9). This is sufficient for psycholinguistic corpora at MECO scale, but is far slower than common inference methods. The unbiased estimator of Chapter 5 likewise estimates prefix probabilities along a given target sequence and does not currently return the full next-symbol distribution at comparable cost. Better batching of source-LM calls and less back and forth between the CPU-heavy transducer operations and GPU inference calls could improve throughput.

7.4 Future directions

Stochastic transduced language models. The construction here treats the transformation f as a function, so each source string has a single image, though the transducer realizing the function may well be nondeterministic as a machine. The natural generalization drops functionality: replace f with a *stochastic map* encoded by a weighted finite-state transducer (WFST), so that the same source string can map to multiple target strings with weights summing to one. The precover, quotient, and remainder objects of Chapter 3 should generalize to weighted analogues.

Training through transducers. The framework currently composes a pretrained, fixed $p_{\mathcal{X}}$ with f at inference time. Instead, the source model could be trained on target-side data by backpropagating through the sum over the preimage of each observed target string. This would extend TLMs from inference-time transformation to learning under a target-side objective.

Cross-vocabulary model combination. Many methods that combine language models through their next-token distributions assume a shared vocabulary, including contrastive decoding (Li et al., 2023), token-level knowledge distillation (Hinton, Vinyals, and Dean, 2015), speculative decoding (Chen et al., 2023; Leviathan, Kalman, and Matias, 2023), and ensembling (Liu et al., 2021). Models from different families often use different vocabularies, so their next-token distributions are not directly comparable. Sections 1.2.2 and 3.5 discuss byte-level conversion and its use in model ensembles. The TLM framework provides a general way to define the models over a common target alphabet.

7.5 Closing remark

The methodological point that runs through the thesis is that a language model's distribution can be a starting point rather than a final commitment. A string drawn from the model and pushed through a transformation is a draw from a new distribution. This thesis has shown that this distribution is a language model in its own right and that its full autoregressive interface can be computed or estimated without updating the source model's parameters. Whenever researchers can choose the units of analysis (as they do in psycholinguistics, computational biology, or in word-level perplexity comparisons), the recommendation is to do so deliberately and convert the model onto those units, rather than inheriting them from a tokenizer.

Appendix A

Inference: algorithms and experiments

This appendix contains material reproduced and adapted from Snæbjarnarson et al. (2026a), with editorial changes for narrative consistency.

This appendix contains additional algorithmic material for Chapter 4, the experimental setup behind Section 4.4, and the full quantitative results summarized there.

A.1 Additional algorithmic material

This section presents the combined-universality shortcut and the all-universal fast path, the backtracking procedure used when pruning reaches a dead end (Section 4.3.4) and a state-based algorithm for decomposing the precover.

A.1.1 Combined universality

Given $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})$ and $\widehat{y} \in \mathcal{Y}$, we want to determine whether $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}\widehat{y})$ using the covering frontier at $\boldsymbol{y}$ without running a full powerset BFS. The frontier $\mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$ contains *committed* elements $(s, \boldsymbol{b})$ with $\boldsymbol{b} \succeq \boldsymbol{y}\widehat{y}$ (the accumulated output already extends past $\boldsymbol{y}$ with symbol $\widehat{y}$) and *boundary* elements $(s, \boldsymbol{b})$ with $\boldsymbol{b} = \boldsymbol{y}$ (the next output symbol is undetermined). The committed states alone may not be universal: some input symbols may lack outgoing transitions from committed states. Boundary states can fill the gaps, provided every transition from a boundary state produces $\widehat{y}$ as its next output symbol.

Proposition A.1 (Combined universality)**.** *Let* $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})$, $\widehat{y} \in \mathcal{Y}$, *and define*

$$S_c = \{s \mid (s, \boldsymbol{b}) \in \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x}),\ \boldsymbol{b} \succeq \boldsymbol{y}\widehat{y}\},$$
$$S_b = \{s \mid (s, \boldsymbol{b}) \in \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x}),\ \boldsymbol{b} = \boldsymbol{y}\}$$

If the union $S_c \cup S_b$ *is universal (i.e.,* $[\![\mathsf{proj}_{\mathcal{X}}(\mathtt{f}_{[S_c \cup S_b]})]\!] = \mathcal{X}^*$*),* $S_c \cap \mathsf{F} \neq \emptyset$*, and every transition from a boundary state produces* $\widehat{y}$*, except that* ε*-input transitions may produce* ε *(i.e.,* $s \in S_b$ *and* $s \xrightarrow{x:y} s' \in \mathsf{T}$ *implies* $y = \widehat{y}$ *or* $(x = \varepsilon \wedge y = \varepsilon)$*), then* $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}\widehat{y})$.

Proof. Let $\boldsymbol{x}' \in \mathcal{X}^*$ be arbitrary. If $\boldsymbol{x}' = \varepsilon$, choose $(s, \boldsymbol{b}) \in \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$ with $s \in S_c \cap \mathsf{F}$ and $\boldsymbol{b} \succeq \boldsymbol{y}\widehat{y}$. The corresponding path is accepting, giving $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y}\widehat{y})$. It remains to consider $\boldsymbol{x}' \neq \varepsilon$. By universality of $S_c \cup S_b$, there exists $s \in S_c \cup S_b$ such that s has an accepting run on $\boldsymbol{x}'$ in the input projection. If $s \in S_c$, choose $(s, \boldsymbol{b}) \in \mathtt{run}_{\boldsymbol{y}}(\boldsymbol{x})$ with $\boldsymbol{b} \succeq \boldsymbol{y}\widehat{y}$. Then the output of the corresponding path in $\mathtt{f}$ already begins with $\boldsymbol{y}\widehat{y}$, giving $\boldsymbol{x}\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y}\widehat{y})$. Otherwise $s \in S_b$, and we take $(s, \boldsymbol{b}) = (s, \boldsymbol{y})$ from the frontier. By the exclusivity hypothesis, every non-ε-input transition from s produces output $\widehat{y}$, and ε-input transitions produce $\widehat{y}$ or ε. States reachable from s via $\varepsilon{:}\varepsilon$ arcs are also in S_b (since the frontier is ε-closed and the buffer is unchanged), so they satisfy the same constraint. Therefore the first non-ε output along the chosen accepting path is $\widehat{y}$ (it exists because $\boldsymbol{x}' \neq \varepsilon$), and the output begins with $\boldsymbol{y}\widehat{y}$, giving $\boldsymbol{x}\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y}\widehat{y})$. Since $\boldsymbol{x}'$ was arbitrary, $\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y}\widehat{y})$, so $\boldsymbol{x} \in \mathcal{C}(\boldsymbol{y}\widehat{y})$. Since $\mathcal{C}(\boldsymbol{y}\widehat{y}) \subseteq \mathcal{C}(\boldsymbol{y})$ and $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})$, $\boldsymbol{x}$ is prefix-minimal in $\mathcal{C}(\boldsymbol{y}\widehat{y})$. Hence $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}\widehat{y})$. ■

This shortcut complements the IP-universality shortcut (Proposition 4.3). IP-universality is a per-state property, whereas combined universality is a set-level property that combines partial coverage from committed and boundary states.

The proposition requires set-level universality of $S_c \cup S_b$ (the input-projection NFA started from those states accepts $\mathcal{X}^*$). `combined_universal` (Algorithm A.1) thus tests a stronger but cheaper sufficient condition, requiring every individual state in $S_c \cup S_b$ to be in the precomputed U. This runs in $O(|\mathcal{F}|)$ time, whereas the set-level check requires a powerset BFS that may be as expensive as the full `is_cylinder` check this shortcut is meant to avoid.

```
def combined_universal(F, y·ŷ):
    S_c ← {s | (s, b) ∈ F, b ⪰ y·ŷ}    # covered
    S_b ← {s | (s, b) ∈ F, b = y}      # boundary
    if ∃ s ∈ S_c ∪ S_b: s ∉ U: return False
    # Non-ε input must produce ŷ; ε input may produce ε
    for s ∈ S_b:
      for (_ --x:y--> s') ∈ T(s):
        if not (y = ŷ or (x = ε ∧ y = ε)):
          return False
    return S_c ∩ F ≠ ∅
```

Algorithm A.1: `combined_universal` replaces the set-level universality condition in Proposition A.1 with the sufficient per-state IP-universality test, making the check linear in the frontier size.

A.1.2 All-universal fast path

Under the conditions of Section 4.3.9, decomposition and scoring simplify as follows. The resulting pseudocode is given in Algorithm A.2. Normalization is unchanged.

Decompose (simplified). Because every state is IP-universal, every source prefix in the precover is a cylinder, so $\mathcal{R}(\boldsymbol{y}) = \emptyset$ for all $\boldsymbol{y}$. Under the additional condition that every non-ε-input arc produces at least one output symbol, every newly discovered live extension is immediately classified as a quotient element, so `decompose`$(\boldsymbol{y})$ (Algorithm 4.2) requires at most one source-symbol expansion. Under these conditions, `decompose_next` is unnecessary because the scoring step below performs the per-symbol classification directly.

Score (simplified). For each $\boldsymbol{x} \in$ `Q`, let $\mathcal{F} = \texttt{run}_{\boldsymbol{y}}(\boldsymbol{x})$. The frontier contains *committed* elements $(s, \boldsymbol{b})$ with $\boldsymbol{b} \succ \boldsymbol{y}$ and *boundary* elements with $\boldsymbol{b} = \boldsymbol{y}$, as in the combined universality shortcut (Section A.1.1). Committed elements have already determined their next output symbol $\widehat{y} = \boldsymbol{b}_{|\boldsymbol{y}|+1}$. By functionality, all committed elements determine the same symbol. Because all states are IP-universal, the quotient prefix contributes $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ once directly to $\overline{p}[\widehat{y}]$, so no expansion or boundary processing is needed.

If there are no committed elements, boundary elements ($\boldsymbol{b} = \boldsymbol{y}$) need one more source symbol to commit to an output. The no-ε-output precondition ensures that each non-ε-input arc produces at least one output symbol, so a precomputed *first-output table* `fo`(S, x) (defined in the pseudocode) maps

each input symbol to the unique output it produces from a given powerstate. This symbol is unique: functionality of f requires that all transitions on x from states in S produce the same output, since IP-universality guarantees each successor has an accepting continuation (so both outputs would be realized). Each boundary element is thus resolved with a single batched LM call: $\ell = \overrightarrow{p_{\mathcal{X}}}(\cdot \mid \boldsymbol{x})$ is queried once per quotient element, and for each $x \in \mathcal{X}$, the mass $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \cdot \ell(x)$ is attributed to $\texttt{fo}(S, x)$. If any boundary state is accepting, the LM's EOS probability contributes $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \cdot \ell(\text{EOS})$ to $\overline{p}[\text{EOS}]$.

A.1.3 Backtracking

Pruning may occasionally lead to dead ends where no valid extension remains. When this occurs while scoring an observed sequence, the next symbol y receives zero conditional probability under the approximate $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$. We attempt to recover using the procedure in Algorithm A.3, which takes the current target string $\boldsymbol{y}$, the next symbol y, the current pruning threshold τ, and hyperparameters: maximum retries R, threshold floor $\tau_{\min}$, and maximum eviction depth $D_{\max}$. On each retry, τ is halved (floored at $\tau_{\min}$ to prevent unlimited $\mathcal{P}$ growth), and cached decompositions for prefixes of $\boldsymbol{y}$ are evicted at exponentially increasing depth $D = \min(2^{i-1}, D_{\max})$, forcing `decompose` to recompute with the relaxed threshold. The original threshold is restored afterward. The procedure is heuristic: a fixed number of retries with halved thresholds may fail to recover. Frequent backtracking signals that the pruning threshold is overly aggressive. The overhead of repeated eviction and recomputation can then dominate runtime.

A.1.4 State-based decomposition

The frontier-based algorithms of Section 4.3 enumerate source *strings*, representing the quotient $\mathcal{Q}(\boldsymbol{y})$ and remainder $\mathcal{R}(\boldsymbol{y})$ as explicit sets. A complementary algorithm operates on the *state space* of the precover DFA rather than its string space. Since $\texttt{P}_{\boldsymbol{y}}$ (Section 4.2.1) has finitely many states, a state-based traversal always terminates, even when either $\mathcal{Q}(\boldsymbol{y})$ or $\mathcal{R}(\boldsymbol{y})$ is infinite. It represents these sets as DFAs `Q` and `R` that *accept* the possibly infinite sets rather than enumerating their elements.

The algorithm in Algorithm A.4 performs a BFS over the states of $\texttt{P}_{\boldsymbol{y}}$. At each state s, it checks whether s is *universal*, i.e., whether the sub-automaton rooted at s accepts $\mathcal{X}^*$ (see `is_cylinder`). If so, s is marked as a quotient member and accepting state, and its successors are not explored, since all extensions from a universal state are covered. If s is accepting but not universal, it is marked as a remainder member and accepting state, and

```
def all_universal_next_dist(y):
  (Q, _) ← decompose(y) # R = ∅ always
  p̄ ← {}; Z ← prefix_prob(y)
  if Z = 0: return {EOS : 1}
  for x in Q:
    F ← run_y(x)
    # Committed: output past target
    C ← {b_{|y|+1} | (s, b) ∈ F, b ≻ y}
    if C ≠ ∅:
      ŷ ← C.pop()  # single element by functionality
      p̄[ŷ] += p→_X(x)
      continue
    # Boundary: output = target
    S_b ← {s | (s, b) ∈ F, b = y}
    if S_b ≠ ∅:
      ℓ ← p→_X(· | x)      # one LM call
      for x ∈ X:
        ŷ ← fo(S_b, x)
        if ŷ ≠ ⊥:
          p̄[ŷ] += p→_X(x) · ℓ(x)
      if ∃ s ∈ S_b: s ∈ F:
        p̄[EOS] += p→_X(x) · ℓ(EOS)
  return p̄/Z

def fo(S, x):
  for (_ --x:y--> s') ∈ T(S, x):
    # Unique if functional &
    # all states IP-universal
    return y
  return ⊥
```

Algorithm A.2: `all_universal_next_dist` specializes `next_dist` (Algorithm 4.10) to transducers whose states are all IP-universal and whose non-ε-input arcs have nonempty output. Every prefix decomposition has $\mathsf{R} = \emptyset$. Boundary elements are resolved via `fo`, requiring at most one LM call per quotient element.

exploration continues through its outgoing arcs. The full precover is then recovered as $\mathcal{P}(\boldsymbol{y}) = \langle[\![\mathsf{Q}]\!]\rangle \sqcup [\![\mathsf{R}]\!]$.

The result is a pair of DFAs that share the same state space and truncated

```
def backtrack(y, y, τ, R, τ_min, D_max):
  τ_0 ← τ
  for i ∈ 1, ... , R:
    τ ← max(τ/2, τ_min)
    D ← min(2^{i−1}, D_max)
    for d ∈ 0, ... , D−1:
      if d > |y|: break
      evict cached decompose(y_{<|y|−d})
    p̄ ← next_dist(y)
    if p̄[y] > 0:
      τ ← τ_0
      return p̄
  τ ← τ_0
  raise DeadEnd
```

Algorithm A.3: `backtrack` retries a zero-probability next symbol by halving τ down to $\tau_{\min}$ and evicting cached decompositions to increasing depth. It restores the original threshold afterward.

arc set but differ in their accepting states. The quotient automaton `Q` has the universal states as accepting states, and arcs leaving universal states are dropped (since the BFS does not expand past them), so $[\![\mathtt{Q}]\!] = \mathcal{Q}(\boldsymbol{y})$, accepting exactly the quotient elements. The remainder automaton `R` uses the same truncated arc set but marks only the non-universal accepting states, so $[\![\mathtt{R}]\!] = \mathcal{R}(\boldsymbol{y})$.

What this buys, and what it doesn't. The state-based algorithm always terminates, even when the string-enumeration algorithms above would need pruning, and the resulting DFAs are compact, finite representations of potentially infinite quotient and remainder sets. The DFA construction does not by itself provide finite-time scoring. Realizing the autoregressive interface (Section 4.1) still requires summing $p_{\mathcal{X}}(\boldsymbol{x})$ over elements of $\mathcal{Q}(\boldsymbol{y})$ and $\mathcal{R}(\boldsymbol{y})$ via Equation (3.8). When these sets are infinite, the DFA representation does not reduce that sum to a finite enumeration.

A.2 Experimental setup

Here we detail the experimental setup for reproducing the experiments in Section 4.4 and Section A.3.5.

```
def dfa_decomposition(f, y):
  P_y ←
    trim(determinize(proj_X(f ∘ yY*)))
  P ← QUEUE(I_y)
  V ← ∅; arcs ← ∅
  Q ← ∅; R ← ∅
  while P:
    s ← P.pop()
    if s ∈ V: continue
    V.add(s)
    if s ∈ F_y:
      if is_cylinder(s):
        Q.add(s)
        continue  # do not expand
      else:
        R.add(s)
    for x ∈ X:
      s' ← step_y(s, x)
      if s' = ∅: continue
      P.push(s')
      arcs.add(s --x--> s')
  return DFA(V,arcs,Q), DFA(V,arcs,R)
```

```
def is_cylinder(s):
  V ← {s}; P ← QUEUE({s})
  while P:
    s ← P.pop()
    if s ∉ F_y:
      return False
    for x ∈ X:
      s' ← step_y(s, x)
      if s' = ∅: return False
      if s' ∉ V:
        V.add(s'); P.push(s')
  return True
```

Algorithm A.4: State-based decomposition of the precover into quotient and remainder DFAs. *Left*: a BFS marks universal states as accepting in `Q` and non-universal accepting states as accepting in `R`, without exploring beyond universal states. *Right*: `is_cylinder` runs the universality BFS directly from a precover-DFA state.

A.2.1 Datasets

The datasets are those used in Section 4.4. The token-to-byte and PTB experiments use WikiText-2 test paragraphs, while the DNA experiments use 65 proteins. The ablation in Section A.3.5 uses the first 256 bytes of each of the first five WikiText-2 test paragraphs. The proteins span 4–12 amino acids each, with UniProt[1] accession numbers in Table A.1.

[1]See `https://www.uniprot.org/`.

Table A.1: Accession numbers of the 65 proteins used in the DNA experiments (Section 4.4).

C0HLZ5	P01858	P0DPI4	P0DUS0	P84464
P84465	P0DOY5	P67857	P67858	P67859
P81826	P23210	P85003	P84071	P86168
C0HJF1	C0HJG0	P02729	P81010	P86909
P86922	B3EWE5	P0DMM6	P0DQM6	P0DQM7
P12481	P85002	B3EWR3	P01358	P02728
P0C005	P0DKX2	P0DMM7	P0DQM9	P0DX30
P22103	P84200	P84785	P84868	P86600
A8C8X2	B3A0L6	B3EUR5	C0HJM6	C0HLK7
P0DJC3	P0DJF4	P69208	P85444	P85870
P86942	A0A0A0MT89	B3EWS0	C0HJB6	C0HL84
C0HL88	P0C8I8	P0DQH7	P0DQH8	P0DQX4
P0DQX5	P58805	P69437	P82820	P83127

A.2.2 Models

We conduct experiments using GPT-2 Large (Radford et al., 2019), Llama 3.2-1B, Llama 3.1-8B and Phi-4 (14B; Abdin et al., 2024)[2] from the 🤗 Hugging Face hub (Wolf et al., 2020). We use the GenLM library[3] and the vLLM (Kwon et al., 2023) backend to evaluate the models efficiently.

For the PTB experiments we use GenLM.bytes[4] to convert token-level models into byte-level models and compose them with $\mathtt{f}_{\text{ptb}}$. For GenLM.bytes we use a beam size of K=5 and a pruning threshold of 0.001. For the DNA experiments we train a custom GPT-2 Small model[5] on a human DNA dataset[6]. The token set of the model is $\mathcal{X} = \{\mathtt{A}, \mathtt{C}, \mathtt{G}, \mathtt{T}\}$, so we need not compose the model with a transducer $\mathtt{f}_{\alpha}$ that maps subword tokens to bytes. For training parameters, see Table A.2, for training and validation metrics Table A.3.

[2] Available under `openai/gpt2-large`, `meta-llama/Llama-3.2-1B`, `meta-llama/Llama-3.1-8B` and microsoft/phi-4 at `https://huggingface.co`.
[3] `https://github.com/genlm/genlm-backend`
[4] `https://github.com/genlm/genlm-bytes`
[5] `https://huggingface.co/vesteinn/gpt2-dna`
[6] `https://huggingface.co/datasets/simecek/Human_DNA_v0`.

A.2.3 Parameters

For all experiments, we use the pruning heuristic described in Section 4.3.4. For the experiments in Section 4.4 we report the results for different values of $n_{\mathsf{max}} \in \{5000, 10000, 15000, 20000, 25000, 30000\}$. For all other experiments, we set $n_{\mathsf{max}} = \infty$. Backtracking (Algorithm A.3) uses $R = 20$, $\tau_{\min} = 10^{-10}$, $D_{\max} = 32$. It is infrequent: for $\tau \leq$ 1e-3, zero fallbacks occur across all experiments. At the coarsest threshold ($\tau =$ 1e-1), the mean number of fallbacks per paragraph reaches at most 96 for $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha}$ (max 159), 5.1 for $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$ (max 18), and 0.4 for $p_{\mathsf{dna}} \circ \mathtt{f}_{\mathrm{dna2aa}}$ (max 5).

A.2.4 GPU usage

All experiments in Section 4.4 use a single NVIDIA GeForce RTX 4090 (24 GB), except Phi-4, which requires two. The benchmarks in Section A.3.5 use an NVIDIA GeForce RTX 3090.

Table A.2: Training parameters for the GPT-2 Small model, trained on human DNA.

Parameter	Value
Learning Rate	0.0003
Optimizer	AdamW $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e{-}8$
Learning Rate Scheduler	Linear
Warm-up Steps	1000
Train Batch Size (per device)	64
Eval Batch Size (per device)	8
Global Train Batch Size	256 (262,144 tokens)
Global Eval Batch Size	32
Epochs	10
Seed	42
Distributed Training	Multi-GPU (4 devices)
Mixed Precision	Native AMP

Table A.3: Training and validation metrics for the GPT-2 Small model, trained on human DNA.

Step	Epoch	Train Loss	Val Loss	Acc (%)
5k	0.69	1.1252	1.1206	47.45
10k	1.38	1.0835	1.0814	49.91
15k	2.07	1.0641	1.0639	51.03
20k	2.76	1.0563	1.0547	51.63
25k	3.45	1.0504	1.0486	52.04
30k	4.14	1.0439	1.0439	52.33
35k	4.84	1.0425	1.0407	52.54
40k	5.53	1.0365	1.0380	52.71
45k	6.22	1.0325	1.0361	52.84
50k	6.91	1.0322	1.0341	52.96
55k	7.60	1.0307	1.0328	53.05
60k	8.29	1.0267	1.0316	53.13
65k	8.98	1.0273	1.0306	53.20
70k	9.67	1.0270	1.0299	53.24

A.2.5 Details on transducers used in experiments

Table A.4 contains the number of states, IP-universal states, and transitions for the transducers described in Section 4.4. We construct all finite-state transducers using Pynini (Gorman, 2016). Note that for experiments using the Penn Treebank FST ($\mathtt{f}_{\mathrm{ptb}}$), we realize $p_{\mathcal{X}} \circ \mathtt{f}_{\alpha}$ using GenLM.bytes, thereby keeping the number of states and arcs constant.

Table A.4: Number of states, IP-universal states, and transitions for each transducer composition used in the experiments of Section 4.4.

Composition	States	IP-Univ. States	Transitions
Tokens to Bytes			
$p_{\text{gpt2}} \circ \mathtt{f}_{\alpha}$	75,723	75,723	125,979
$p_{\text{llama1B}} \circ \mathtt{f}_{\alpha}$	176,990	176,990	305,244
$p_{\text{llama8B}} \circ \mathtt{f}_{\alpha}$	176,990	176,990	305,244
$p_{\text{phi4}} \circ \mathtt{f}_{\alpha}$	115,244	115,244	215,593
Tokens to Words			
$p_{\text{gpt2}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\text{ptb}}$	479	56	26,962
$p_{\text{llama1B}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\text{ptb}}$	479	56	26,962
$p_{\text{llama8B}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\text{ptb}}$	479	56	26,962
$p_{\text{phi4}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\text{ptb}}$	479	56	26,962
DNA to amino acids			
$p_{\text{dna}} \circ \mathtt{f}_{\text{dna2aa}}$	21	21	84

Constructing the PTB transducer. We construct the PTB FST by encoding each tokenizer rule[7] as an FST that segments character sequences by inserting a distinguished separator symbol SEP $\notin \mathcal{Y}$. Specifically, the transducer inserts SEP at punctuation and clitic boundaries, and collapses whitespace characters (spaces, tabs, newlines, etc.) into SEP. Note that the resulting transduced language model is thus not a true distribution over PTB tokens, but over characters and separators corresponding to the same boundaries that the PTB tokens would have. This is a pragmatic decision, as the PTB tokenizer can tokenize any sentence into orthographic words. In other words, it would accept an infinite vocabulary. Such a transducer can be built on the fly and would be equivalent to one with infinitely many states. In this thesis, we only use the finite version. An example of such a rule is given in Figure 4.2, which inserts SEP before a comma if it is not followed by a digit. We then compose these FSTs into a single transducer ($\mathtt{f}_{\text{ptb}}$). Note that context-dependent rules, such as the one given in Figure 4.2, introduce non-IP-universal states. For example, state q_2 accepts only $x \in \mathcal{X} \setminus \{\texttt{0--9,}\}$.

[7]The full specification is available at `https://www.nltk.org/_modules/nltk/tokenize/treebank.html#TreebankWordTokenizer`.

In fact, of the 479 states in $\mathtt{f}_{\text{ptb}}$, just 56 are IP-universal.

The DNA-to-amino-acid transducer. The DNA-to-amino-acid transducer, described in Section 4.4, is partially shown in Figure A.1.

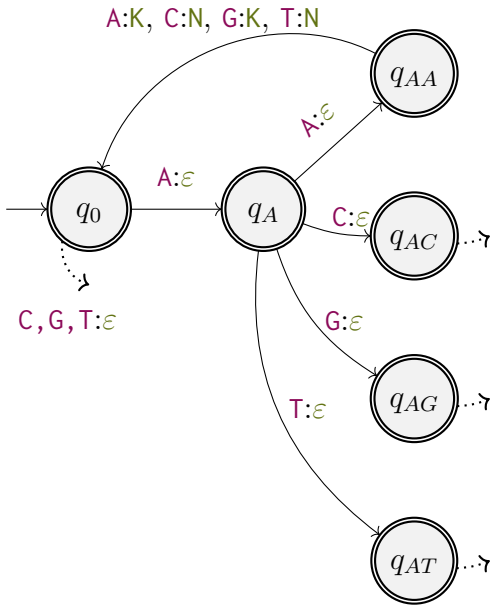


Figure A.1: Part of an FST that maps each DNA codon to one of 20 amino-acid symbols or the stop symbol *.

A.3 Additional experimental results

Here we provide complementary results to the experiments presented in Section 4.4, using the algorithms described in Section 4.3. In Section A.3.1, we compare our transducer-based method for $\mathtt{f}_\alpha$ with GenLM.bytes (Vieira et al., 2025a). We then give full JSD comparisons for $\mathtt{f}_\alpha$, $\mathtt{f}_{\text{ptb}}$, and $\mathtt{f}_{\text{dna2aa}}$ in Section A.3.2. In Section A.3.4, we report cross-entropy comparisons for $\mathtt{f}_\alpha$. Finally, Section A.3.5 assesses the importance of IP-universal states by ablation.

A.3.1 Comparison to Vieira et al. (2025a)

Vieira et al. (2025a) convert token-level models to byte-level models using a beam-search algorithm parameterized by a beam size K. We compare our transducer-based approach with their method by computing the JSD between the two resulting distributions over the paragraphs from WikiText

(Section A.2). Table A.6 uses genlm-bytes ($K = 32$, pruning threshold of 0.0) as a reference and varies our pruning threshold τ. As τ decreases, JSD drops, and our distributions converge to the baseline. At moderate thresholds ($\tau \leq$ 1e-3), JSD is already below 1.5e-3 for all models, confirming that the two methods produce closely matching distributions.

To verify that the two methods compute the same underlying distribution, we use $\tau = 0$ for our method and exact inference for genlm-bytes ($K = \infty$). Table A.5 shows the per-position JSD and maximum absolute difference for the first 10 bytes of the WikiText evaluation data using $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha}$. The JSD values are on the order of 1e-14–1e-15, and the maximum absolute probability difference is at most 3.4e-6, confirming that both methods give the same distribution up to small floating-point differences.

Table A.7 reports the JSD and throughput of the baseline itself at $K \in \{8, 16, 32\}$, showing the accuracy–speed trade-off within their method. Reducing K from 32 to 8 substantially increases throughput, while the resulting distributions remain close to those at $K = 32$. The method of Vieira et al. (2025a) is designed specifically for strict-prefix-monotone transformations, a property that $\mathtt{f}_{\alpha}$ satisfies. This specialization enables a trie-based algorithm that achieves a better accuracy–speed trade-off than our more general-purpose approach on this class of transformations.

A.3.2 Jensen–Shannon divergence

We benchmark the effect of probability-mass pruning (Section 4.3.4) by comparing distributions obtained at higher thresholds τ with tighter-threshold reference distributions. Across all three settings (token-to-byte, PTB tokenization, and DNA-to-amino-acid), JSD generally decreases as τ decreases, at the cost of throughput (target symbols/sec).

Table A.8 reports the token-to-byte results ($p_{\mathcal{X}} \circ \mathtt{f}_{\alpha}$) for all four models. JSD drops by two to three orders of magnitude from $\tau =$ 1e-1 to $\tau =$ 3e-5, with the tightest thresholds reaching JSD below 1e-4 for all models.

Table A.9 gives the corresponding results for PTB tokenization ($p_{\mathcal{X}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$), complementing Figure 4.3 (right). Although $\mathtt{f}_{\mathrm{ptb}}$ has fewer states than the token-to-byte transducers, only 56 of its 479 states are IP-universal, making lower thresholds computationally expensive. The reference is therefore $\tau =$ 1e-4 (3e-4 for p_{phi4}) rather than 1e-5. Despite this, JSD converges quickly, reaching the order of 1e-5 at $\tau =$ 3e-4 for p_{gpt2}, p_{llama1B}, and p_{llama8B}, while p_{phi4} converges more slowly.

Tables A.10 and A.11 report the DNA-to-amino-acid results ($p_{\mathsf{dna}} \circ \mathtt{f}_{\mathrm{dna2aa}}$), where we also vary the candidate-set cap n_{max} to mitigate the combinatorial blow-up that arises because many amino acids can each be encoded by multiple

Table A.5: Per-position JSD and maximum absolute probability difference over the first ten WikiText-2 bytes, comparing our unpruned method ($\tau = 0$) with exact genlm-bytes for $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha}$.

Pos	Byte	JSD	max $\lvert\Delta p\rvert$
0	R	3.9e-15	7.9e-7
1	o	6.5e-16	1.2e-6
2	b	3.5e-15	1.2e-6
3	e	4.4e-14	1.7e-6
4	r	4.5e-15	3.4e-6
5	t	1.8e-15	2.4e-6
6	␣	1.1e-14	1.4e-6
7	B	1.9e-15	1.8e-6
8	o	6.1e-15	1.2e-6
9	u	1.3e-14	1.5e-6

codons. Tighter thresholds generally reduce JSD. However, the interaction between τ and n_{max} is less uniform, as both the evaluated and reference distributions depend on the cap.

Incomplete runs. As shown in Section A.3.3, the decomposition size grows rapidly at tight thresholds: mean $\lvert\mathtt{Q}\rvert$ exceeds 35,000 at $\tau =$ 1e-5 for $\mathtt{f}_{\alpha}$. For some model–threshold combinations, this growth prevents calculating the full distribution at each position in all ten evaluation paragraphs. Specifically, $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha}$ in Table A.8 completed only 7 of 10 paragraphs at $\tau =$ 1e-5. In Table A.9, $p_{\mathsf{phi4}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$ completed 7 of 10 paragraphs at its reference threshold of $\tau =$ 3e-4, and $p_{\mathsf{llama1B}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$ completed 9 of 10 at $\tau =$ 1e-4. Additionally, $p_{\mathsf{phi4}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$ completed only 9 of 10 paragraphs at $\tau =$ 1e-1 due to an unrecoverable backtracking failure (Algorithm A.3). In these cases, JSD and throughput are computed over the paragraphs shared across all thresholds for that model–transducer combination. The cross-entropy runs of Section A.3.4 are separate computations with their own completion counts: there, $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha}$ at $\tau =$ 1e-5 completes 6 of 10 paragraphs (Table A.13).

Table A.6: Average Jensen–Shannon divergence (JSD) and bytes/sec for various thresholds τ using $p_{\mathcal{X}} \circ \mathtt{f}_\alpha$ against a reference distribution from Vieira et al. (2025a) with a beam size of K=32. 95% confidence intervals are given in parentheses.

	$p_{\mathsf{gpt2}} \circ \mathtt{f}_\alpha$		$p_{\mathsf{llama1B}} \circ \mathtt{f}_\alpha$	
τ	average JSD / byte	bytes / sec	average JSD / byte	bytes / sec
1e-1	4.0e-2 (3.7e-2, 4.2e-2)	70.95 (64.14, 79.65)	2.8e-2 (2.5e-2, 3.0e-2)	84.21 (71.77, 99.54)
3e-2	1.9e-2 (1.8e-2, 2.1e-2)	44.68 (40.59, 48.66)	1.2e-2 (1.1e-2, 1.4e-2)	72.23 (65.87, 78.10)
1e-2	6.7e-3 (5.8e-3, 7.8e-3)	25.13 (23.83, 26.69)	4.2e-3 (3.5e-3, 4.9e-3)	40.45 (37.60, 43.43)
3e-3	2.5e-3 (1.9e-3, 3.3e-3)	10.48 (9.89, 11.12)	1.1e-3 (8.4e-4, 1.3e-3)	22.09 (20.94, 23.51)
1e-3	1.4e-3 (9.3e-4, 1.9e-3)	6.21 (5.89, 6.59)	4.0e-4 (2.9e-4, 5.6e-4)	11.86 (11.27, 12.49)
3e-4	7.6e-5 (6.0e-5, 9.9e-5)	1.97 (1.84, 2.11)	1.3e-4 (1.1e-4, 1.5e-4)	5.90 (5.60, 6.22)
1e-4	3.7e-5 (3.0e-5, 4.7e-5)	0.65 (0.61, 0.71)	8.8e-5 (8.1e-5, 9.5e-5)	2.15 (2.04, 2.26)
3e-5	2.5e-5 (2.4e-5, 2.7e-5)	0.21 (0.19, 0.22)	7.8e-5 (7.3e-5, 8.3e-5)	1.20 (1.15, 1.27)
1e-5	2.5e-5 (2.3e-5, 2.7e-5)	0.19 (0.17, 0.21)	7.4e-5 (6.9e-5, 7.7e-5)	0.62 (0.59, 0.66)

	$p_{\mathsf{llama8B}} \circ \mathtt{f}_\alpha$		$p_{\mathsf{phi4}} \circ \mathtt{f}_\alpha$	
τ	average JSD / byte	bytes / sec	average JSD / byte	bytes / sec
1e-1	2.2e-2 (2.0e-2, 2.4e-2)	79.03 (71.08, 87.28)	2.5e-2 (2.3e-2, 2.7e-2)	41.04 (36.96, 45.33)
3e-2	8.8e-3 (7.7e-3, 1.0e-2)	66.41 (62.69, 70.69)	9.7e-3 (8.6e-3, 1.1e-2)	36.20 (33.26, 38.87)
1e-2	3.5e-3 (2.9e-3, 4.2e-3)	45.47 (43.10, 47.91)	4.6e-3 (3.8e-3, 5.4e-3)	22.85 (21.14, 24.77)
3e-3	1.3e-3 (9.3e-4, 1.8e-3)	26.71 (25.11, 28.34)	1.8e-3 (1.4e-3, 2.4e-3)	14.80 (13.70, 15.83)
1e-3	5.1e-4 (2.9e-4, 8.4e-4)	15.73 (14.88, 16.60)	8.5e-4 (5.0e-4, 1.3e-3)	8.75 (8.30, 9.27)
3e-4	1.3e-4 (1.1e-4, 1.5e-4)	9.08 (8.68, 9.57)	1.9e-4 (1.4e-4, 2.5e-4)	4.87 (4.60, 5.16)
1e-4	9.1e-5 (8.0e-5, 1.0e-4)	4.03 (3.84, 4.23)	1.0e-4 (8.8e-5, 1.2e-4)	2.19 (2.07, 2.33)
3e-5	7.5e-5 (6.8e-5, 8.3e-5)	2.38 (2.27, 2.49)	7.6e-5 (6.8e-5, 8.4e-5)	1.30 (1.23, 1.37)
1e-5	7.3e-5 (6.6e-5, 8.1e-5)	1.40 (1.33, 1.47)	7.9e-5 (7.2e-5, 8.8e-5)	0.74 (0.70, 0.79)

Table A.8: Average Jensen–Shannon divergence (JSD) and bytes/sec for various thresholds τ using $p_{\mathcal{X}} \circ \mathtt{f}_\alpha$ against a reference distribution (τ =1e-5). 95% confidence intervals are given in parentheses.

	$p_{\mathsf{gpt2}} \circ \mathtt{f}_\alpha$		$p_{\mathsf{llama1B}} \circ \mathtt{f}_\alpha$	
τ	average JSD / byte	bytes / sec	average JSD / byte	bytes / sec
1e-1	3.8e-2 (3.4e-2, 4.1e-2)	76.50 (66.94, 89.01)	2.8e-2 (2.6e-2, 3.0e-2)	84.21 (72.49, 99.24)
3e-2	1.7e-2 (1.5e-2, 1.9e-2)	45.61 (39.82, 51.46)	1.2e-2 (1.1e-2, 1.4e-2)	72.23 (66.25, 78.94)
1e-2	5.9e-3 (4.8e-3, 7.1e-3)	25.71 (24.08, 27.89)	4.2e-3 (3.5e-3, 5.0e-3)	40.45 (37.65, 43.43)
3e-3	2.1e-3 (1.4e-3, 2.9e-3)	11.22 (10.44, 12.07)	1.1e-3 (8.5e-4, 1.3e-3)	22.09 (20.90, 23.34)
1e-3	1.5e-3 (9.1e-4, 2.3e-3)	6.49 (6.02, 6.99)	4.0e-4 (2.8e-4, 5.4e-4)	11.86 (11.25, 12.54)
3e-4	5.5e-5 (4.7e-5, 6.7e-5)	2.28 (2.09, 2.48)	1.2e-4 (1.1e-4, 1.4e-4)	5.90 (5.59, 6.23)
1e-4	3.6e-5 (2.9e-5, 4.8e-5)	1.00 (0.92, 1.09)	7.4e-5 (6.9e-5, 8.0e-5)	2.15 (2.03, 2.26)
3e-5	2.4e-5 (2.2e-5, 2.7e-5)	0.44 (0.40, 0.49)	5.2e-5 (4.9e-5, 5.6e-5)	1.20 (1.14, 1.26)
1e-5	(not applicable)	0.19 (0.17, 0.21)	(not applicable)	0.62 (0.59, 0.65)

	$p_{\mathsf{llama8B}} \circ \mathtt{f}_\alpha$		$p_{\mathsf{phi4}} \circ \mathtt{f}_\alpha$	
τ	average JSD / byte	bytes / sec[123]	average JSD / byte	bytes / sec
1e-1	2.3e-2 (2.1e-2, 2.5e-2)	79.03 (71.62, 87.24)	2.5e-2 (2.3e-2, 2.7e-2)	41.04 (36.75, 45.30)
3e-2	8.9e-3 (7.8e-3, 1.0e-2)	66.41 (61.96, 70.51)	9.9e-3 (8.7e-3, 1.1e-2)	36.20 (33.36, 38.85)
1e-2	3.5e-3 (2.9e-3, 4.4e-3)	45.47 (43.16, 48.08)	4.7e-3 (3.8e-3, 5.6e-3)	22.85 (21.06, 24.55)
3e-3	1.3e-3 (9.4e-4, 1.8e-3)	26.71 (25.17, 28.24)	1.8e-3 (1.3e-3, 2.4e-3)	14.80 (13.74, 15.78)
1e-3	5.2e-4 (3.0e-4, 8.4e-4)	15.73 (14.89, 16.61)	8.3e-4 (4.8e-4, 1.2e-3)	8.75 (8.30, 9.27)
3e-4	1.2e-4 (1.1e-4, 1.4e-4)	9.08 (8.64, 9.60)	1.9e-4 (1.1e-4, 2.9e-4)	4.87 (4.61, 5.19)

Table A.7: Throughput and Jensen–Shannon divergence (JSD) of Vieira et al. (2025a) at beam sizes $K \in \{8, 16, 32\}$. JSD is measured against $K = 32$. 95% confidence intervals are given in parentheses.

	$p_{\mathsf{gpt2}} \circ \mathsf{f}_\alpha$		$p_{\mathsf{llama1B}} \circ \mathsf{f}_\alpha$	
K	JSD vs $K=32$	bytes / sec	JSD vs $K=32$	bytes / sec
8	3.0e-5 (2.5e-5, 3.9e-5)	37.81 (37.59, 37.98)	7.1e-5 (6.7e-5, 7.6e-5)	46.94 (45.69, 47.97)
16	2.4e-5 (2.3e-5, 2.7e-5)	28.83 (28.53, 29.13)	7.1e-5 (6.7e-5, 7.5e-5)	28.76 (28.42, 29.08)
32	(not applicable)	18.12 (17.87, 18.36)	(not applicable)	5.34 (4.21, 7.19)
	$p_{\mathsf{llama8B}} \circ \mathsf{f}_\alpha$		$p_{\mathsf{phi4}} \circ \mathsf{f}_\alpha$	
K	JSD vs $K=32$	bytes / sec	JSD vs $K=32$	bytes / sec
8	6.8e-5 (6.3e-5, 7.4e-5)	18.30 (18.17, 18.43)	4.6e-5 (4.3e-5, 4.9e-5)	11.59 (11.29, 11.80)
16	6.9e-5 (6.2e-5, 7.8e-5)	10.55 (10.45, 10.65)	5.3e-5 (4.9e-5, 5.6e-5)	7.90 (7.84, 7.96)
32	(not applicable)	5.85 (5.79, 5.91)	(not applicable)	4.51 (4.44, 4.56)

Table A.9: Average Jensen–Shannon divergence (JSD) and bytes/sec for various thresholds τ using $p_{\mathcal{X}} \circ \mathsf{f}_\alpha \circ \mathsf{f}_{\mathrm{ptb}}$ against a reference distribution ($\tau =$ 1e-4; 3e-4 for p_{phi4}). 95% confidence intervals are given in parentheses.

	$p_{\mathsf{gpt2}} \circ \mathsf{f}_\alpha \circ \mathsf{f}_{\mathrm{ptb}}$		$p_{\mathsf{llama1B}} \circ \mathsf{f}_\alpha \circ \mathsf{f}_{\mathrm{ptb}}$	
τ	average JSD / byte	bytes / sec	average JSD / byte	bytes / sec
1e-1	3.9e-3 (3.3e-3, 4.7e-3)	26.65 (25.79, 27.48)	3.0e-3 (2.6e-3, 3.5e-3)	30.44 (28.31, 32.19)
3e-2	1.4e-3 (1.2e-3, 1.5e-3)	17.03 (16.42, 17.66)	1.1e-3 (9.3e-4, 1.3e-3)	17.45 (16.67, 18.27)
1e-2	5.7e-4 (5.0e-4, 6.7e-4)	10.79 (10.23, 11.36)	3.7e-4 (3.4e-4, 4.0e-4)	10.54 (10.03, 11.09)
3e-3	1.7e-4 (1.5e-4, 1.9e-4)	4.52 (4.29, 4.79)	1.3e-4 (1.2e-4, 1.4e-4)	8.30 (7.98, 8.67)
1e-3	5.5e-5 (5.1e-5, 6.1e-5)	2.01 (1.88, 2.15)	6.1e-5 (5.8e-5, 6.5e-5)	4.13 (3.91, 4.36)
3e-4	2.0e-5 (1.7e-5, 2.4e-5)	0.81 (0.77, 0.85)	3.3e-5 (3.1e-5, 3.5e-5)	1.67 (1.59, 1.76)
1e-4	(not applicable)	0.20 (0.16, 0.25)	(not applicable)	0.70 (0.66, 0.74)
	$p_{\mathsf{llama8B}} \circ \mathsf{f}_\alpha \circ \mathsf{f}_{\mathrm{ptb}}$		$p_{\mathsf{phi4}} \circ \mathsf{f}_\alpha \circ \mathsf{f}_{\mathrm{ptb}}$	
τ	average JSD / byte	bytes / sec	average JSD / byte	bytes / sec
1e-1	2.5e-3 (2.1e-3, 3.0e-3)	19.25 (18.79, 19.76)	4.6e-3 (3.2e-3, 6.5e-3)	10.07 (9.53, 10.67)
3e-2	1.0e-3 (8.6e-4, 1.2e-3)	13.38 (12.99, 13.83)	3.8e-3 (2.6e-3, 5.1e-3)	5.87 (5.37, 6.40)
1e-2	6.0e-4 (4.7e-4, 7.8e-4)	9.49 (9.16, 9.88)	3.2e-3 (2.2e-3, 4.4e-3)	2.94 (2.59, 3.35)
3e-3	1.1e-4 (1.0e-4, 1.2e-4)	5.90 (5.62, 6.20)	4.2e-4 (3.0e-4, 5.8e-4)	2.95 (2.68, 3.26)
1e-3	5.1e-5 (4.7e-5, 5.5e-5)	3.29 (3.13, 3.45)	3.6e-4 (2.5e-4, 4.8e-4)	1.54 (1.37, 1.74)
3e-4	2.3e-5 (2.1e-5, 2.5e-5)	1.42 (1.35, 1.49)	(not applicable)	0.48 (0.43, 0.54)
1e-4	(not applicable)	0.56 (0.53, 0.59)	(not applicable)	(not applicable)

Table A.10: Average Jensen–Shannon divergence (JSD) and amino acids/sec for various thresholds τ and a reference (τ =1e-6) using $p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$, for candidate-set caps $n_{\mathsf{max}} \in \{5000, 10000, 15000, 20000\}$. 95% confidence intervals are given in parentheses. We limit the candidate-set size (n_{max}) to mitigate the combinatorial blow-up with increasing sequence length, and Table A.11 continues with the larger caps.

	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ ($n_{\mathsf{max}} = 5000$)		$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ ($n_{\mathsf{max}} = 10000$)	
τ	average JSD / amino acid	amino acids / sec	average JSD / amino acid	amino acids / sec
1e-1	2.8e-2 (2.4e-2, 3.2e-2)	9.55 (8.44, 10.90)	2.8e-2 (2.5e-2, 3.3e-2)	9.54 (8.45, 10.82)
3e-2	3.2e-3 (2.8e-3, 3.6e-3)	2.90 (2.57, 3.26)	3.3e-3 (2.9e-3, 3.7e-3)	2.03 (1.74, 2.39)
1e-2	6.6e-4 (4.4e-4, 9.0e-4)	2.20 (1.99, 2.47)	7.4e-4 (5.0e-4, 1.0e-3)	1.42 (1.25, 1.63)
3e-3	1.6e-4 (7.8e-5, 2.7e-4)	1.89 (1.72, 2.11)	2.7e-4 (1.1e-4, 4.9e-4)	1.16 (1.03, 1.31)
1e-3	7.1e-5 (1.7e-5, 1.6e-4)	1.79 (1.62, 1.97)	6.6e-5 (1.7e-5, 1.5e-4)	1.08 (0.97, 1.22)
3e-4	4.9e-5 (9.1e-6, 1.1e-4)	1.73 (1.58, 1.93)	3.5e-5 (6.7e-6, 7.9e-5)	1.04 (0.93, 1.17)
1e-4	5.8e-5 (1.1e-5, 1.2e-4)	1.72 (1.56, 1.91)	2.1e-5 (9.9e-7, 7.9e-5)	1.02 (0.92, 1.16)
3e-5	3.0e-5 (7.9e-7, 7.9e-5)	1.71 (1.55, 1.89)	2.1e-5 (6.6e-7, 6.2e-5)	1.02 (0.92, 1.14)
1e-5	2.2e-5 (1.0e-6, 5.4e-5)	1.71 (1.56, 1.89)	3.9e-6 (5.1e-7, 8.9e-6)	1.02 (0.91, 1.15)
3e-6	1.0e-5 (2.3e-7, 2.9e-5)	1.70 (1.55, 1.89)	1.2e-5 (6.1e-7, 3.2e-5)	1.01 (0.91, 1.12)
1e-6	(not applicable)	1.71 (1.55, 1.90)	(not applicable)	1.02 (0.92, 1.14)

	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ ($n_{\mathsf{max}} = 15000$)		$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ ($n_{\mathsf{max}} = 20000$)	
τ	average JSD / amino acid	amino acids / sec	average JSD / amino acid	amino acids / sec
1e-1	2.9e-2 (2.5e-2, 3.3e-2)	9.52 (8.46, 10.82)	2.9e-2 (2.5e-2, 3.3e-2)	9.35 (8.21, 10.65)
3e-2	3.2e-3 (2.8e-3, 3.5e-3)	1.63 (1.37, 1.94)	3.2e-3 (2.8e-3, 3.7e-3)	1.41 (1.18, 1.71)
1e-2	6.8e-4 (4.5e-4, 9.8e-4)	1.10 (0.96, 1.27)	8.3e-4 (5.0e-4, 1.3e-3)	0.91 (0.78, 1.07)
3e-3	2.2e-4 (8.2e-5, 4.2e-4)	0.88 (0.77, 1.00)	2.0e-4 (7.6e-5, 3.9e-4)	0.71 (0.62, 0.83)
1e-3	5.0e-5 (8.5e-6, 1.3e-4)	0.80 (0.71, 0.91)	6.6e-5 (1.7e-5, 1.5e-4)	0.64 (0.56, 0.73)
3e-4	2.8e-5 (3.6e-6, 7.2e-5)	0.77 (0.68, 0.88)	3.0e-5 (3.9e-6, 7.9e-5)	0.61 (0.55, 0.71)
1e-4	2.4e-5 (1.1e-6, 7.0e-5)	0.75 (0.66, 0.84)	6.0e-5 (1.8e-6, 1.6e-4)	0.60 (0.54, 0.69)
3e-5	6.0e-5 (1.0e-6, 1.6e-4)	0.74 (0.66, 0.84)	1.8e-4 (5.0e-5, 3.7e-4)	0.60 (0.53, 0.68)
1e-5	3.0e-7 (7.1e-9, 8.3e-7)	0.74 (0.66, 0.84)	7.8e-7 (5.3e-8, 2.1e-6)	0.59 (0.53, 0.68)
3e-6	9.3e-7 (1.1e-8, 2.7e-6)	0.73 (0.65, 0.83)	7.3e-7 (8.9e-9, 2.1e-6)	0.59 (0.53, 0.67)
1e-6	(not applicable)	0.74 (0.65, 0.84)	(not applicable)	0.60 (0.53, 0.69)

Table A.11: Average Jensen–Shannon divergence (JSD) and amino acids/sec for various thresholds τ and a reference (τ =1e-6) using $p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$, for candidate-set caps $n_{\mathsf{max}} \in \{25000, 30000\}$, continuing Table A.10. 95% confidence intervals are given in parentheses.

τ	$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ ($n_{\mathsf{max}} = 25000$)		$p_{\mathsf{dna}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ ($n_{\mathsf{max}} = 30000$)	
	average JSD / amino acid	amino acids / sec	average JSD / amino acid	amino acids / sec
1e-1	2.9e-2 (2.5e-2, 3.3e-2)	9.50 (8.31, 10.78)	2.8e-2 (2.5e-2, 3.3e-2)	9.52 (8.43, 10.99)
3e-2	3.3e-3 (2.9e-3, 3.7e-3)	1.28 (1.06, 1.60)	3.3e-3 (2.9e-3, 3.8e-3)	1.18 (0.96, 1.45)
1e-2	7.9e-4 (4.9e-4, 1.3e-3)	0.81 (0.70, 0.95)	8.2e-4 (4.9e-4, 1.2e-3)	0.72 (0.61, 0.86)
3e-3	2.3e-4 (8.1e-5, 4.7e-4)	0.62 (0.54, 0.72)	2.5e-4 (7.9e-5, 4.8e-4)	0.55 (0.47, 0.64)
1e-3	5.9e-5 (1.1e-5, 1.4e-4)	0.56 (0.49, 0.65)	6.0e-5 (1.1e-5, 1.4e-4)	0.49 (0.42, 0.57)
3e-4	2.6e-5 (3.5e-6, 6.8e-5)	0.53 (0.46, 0.61)	3.4e-5 (4.9e-6, 8.5e-5)	0.46 (0.41, 0.53)
1e-4	2.4e-5 (8.2e-7, 6.9e-5)	0.52 (0.46, 0.59)	2.4e-5 (1.0e-6, 8.0e-5)	0.45 (0.39, 0.51)
3e-5	7.2e-5 (1.9e-5, 1.5e-4)	0.51 (0.45, 0.59)	2.4e-5 (5.6e-7, 7.1e-5)	0.45 (0.39, 0.52)
1e-5	1.9e-6 (5.6e-8, 4.6e-6)	0.51 (0.45, 0.59)	1.7e-7 (5.7e-9, 3.8e-7)	0.44 (0.39, 0.51)
3e-6	1.3e-6 (1.2e-8, 3.3e-6)	0.51 (0.45, 0.58)	1.6e-6 (1.7e-8, 3.7e-6)	0.44 (0.38, 0.51)
1e-6	(not applicable)	0.51 (0.45, 0.59)	(not applicable)	0.44 (0.39, 0.51)

A.3.3 Decomposition size

Table A.12 and Figure A.2 show how the quotient and remainder sizes change as τ decreases, explaining the throughput reduction observed in the JSD experiments.

For the all-IP-universal token-to-byte and DNA transducers, the remainder is empty ($|\mathsf{R}|=0$) at every position, so only the quotient size is reported. In the token-to-byte setting, the mean quotient size grows from 62 at τ = 1e-1 to over 35,000 at τ = 1e-5, directly accounting for the throughput reduction at tight thresholds. The DNA transducer saturates early due to $n_{\mathsf{max}} = 5000$: mean $|\mathsf{Q}|$ reaches 465 by τ = 3e-5 and remains flat beyond that.

For $\mathsf{f}_{\mathrm{ptb}}$, which has non-IP-universal states, both $|\mathsf{Q}|$ and $|\mathsf{R}|$ are reported. $|\mathsf{Q}|$ grows steadily with a tighter τ, while $|\mathsf{R}|$ initially decreases as more candidates enter the quotient but then grows as the expansion loop over non-IP-universal states discovers additional remainder elements. The presence of a non-trivial remainder is the key structural difference from the all-IP-universal transducers: each remainder element requires a full string probability $p_{\mathcal{X}}(\boldsymbol{x})$ rather than just a prefix probability, making it more expensive per element.

Table A.12: Mean and maximum decomposition sizes across sequence positions for three transducer compositions at varying pruning thresholds τ. Results are computed on paragraph 1 of WikiText (833 bytes for $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha}$, 850 bytes for $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$) and on the longest protein in the test set (P83127, 12 amino acids for $p_{\mathsf{dna}} \circ \mathtt{f}_{\mathrm{dna2aa}}$). The token-to-byte and DNA compositions have empty remainders because all FST states are IP-universal. The PTB composition also reports remainder size.

(a) $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha}$

τ	Mean $\lvert\mathtt{Q}\rvert$	Max $\lvert\mathtt{Q}\rvert$
1e-1	62	2,238
3e-2	200	5,736
1e-2	439	11,998
3e-3	873	18,761
1e-3	1,536	25,341
3e-4	3,161	65,536
1e-4	7,800	131,072
3e-5	17,232	524,288
1e-5	35,768	2,097,152

(b) $p_{\mathsf{dna}} \circ \mathtt{f}_{\mathrm{dna2aa}}$

τ	Mean $\lvert\mathtt{Q}\rvert$	Max $\lvert\mathtt{Q}\rvert$
1e-1	104	344
3e-2	288	1,274
1e-2	346	1,329
3e-3	439	2,011
1e-3	454	2,027
3e-4	462	2,044
1e-4	463	2,044
3e-5	465	2,046
1e-5	465	2,046
3e-6	465	2,046

(c) $p_{\mathsf{gpt2}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$

τ	Mean $\lvert\mathtt{Q}\rvert$	Max $\lvert\mathtt{Q}\rvert$	Mean $\lvert\mathtt{R}\rvert$	Max $\lvert\mathtt{R}\rvert$
1e-1	1.3	31	1.0	113
3e-2	1.9	57	0.38	113
1e-2	3.6	142	0.48	25
3e-3	7.3	353	1.2	53
1e-3	16	686	3.5	129
3e-4	47	1,807	11.4	408
1e-4	128	5,101	33.9	1,195

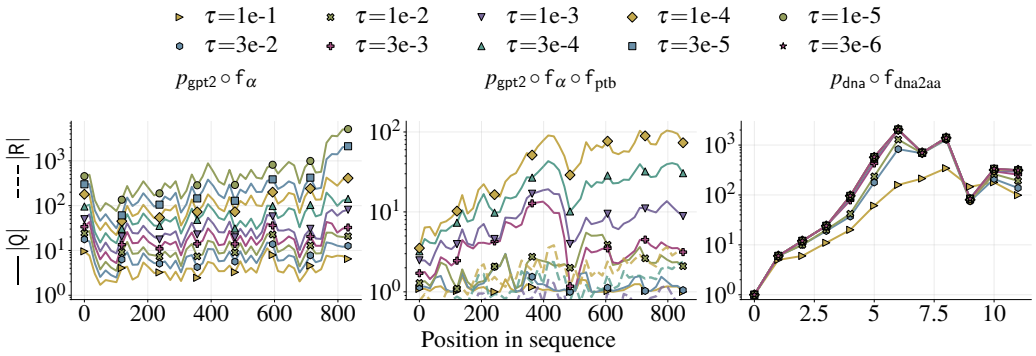


Figure A.2: Quotient size |Q| (solid) and remainder size |R| (dashed) by sequence position and pruning threshold τ. From left to right, the panels show token-to-byte on WikiText paragraph 1 (833 bytes), PTB tokenization of the same paragraph (850 bytes), and DNA-to-amino-acid transduction of the longest test protein (P83127, 12 amino acids). The first and third compositions have empty remainders because all FST states are IP-universal. The DNA panel uses a candidate-set cap of $n_{\mathsf{max}} = 5{,}000$, limiting quotient growth at tighter thresholds.

A.3.4 Cross-entropy and throughput

Cross-entropy measures how well the transduced model assigns probability to held-out text. Unlike JSD, it only requires evaluating $\overrightarrow{p_{\mathcal{Y}}}(y \mid \boldsymbol{y})$ for the observed next symbol rather than computing the full next-symbol distribution $\overrightarrow{p_{\mathcal{Y}}}(\cdot \mid \boldsymbol{y})$. Tables A.13 and A.14 report cross-entropy (in nats and bits per byte) together with throughput for $p_{\mathcal{X}} \circ \mathsf{f}_\alpha$. Because only a single probability is needed per position, throughput is higher than in the JSD tables above. Cross-entropy stabilizes as τ decreases, with little change across the tightest comparable thresholds for all four models.

Table A.13: Cross-entropy in nats per byte and bits per byte, together with throughput, for various thresholds τ using $p_{\mathcal{X}} \circ \mathsf{f}_\alpha$, for p_{gpt2} and p_{llama1B} (Table A.14 continues with p_{llama8B} and p_{phi4}). 95% confidence intervals are given in parentheses. $p_{\mathsf{gpt2}} \circ \mathsf{f}_\alpha$ at $\tau =$ 1e-5 is computed over 6 paragraphs. All other configurations use 10.

τ	bytes / sec		Bits / byte		Cross-entropy	
	Mean	95% CI	Mean	95% CI	Mean	95% CI
			$p_{\mathsf{gpt2}} \circ \mathsf{f}_\alpha$			
1e-1	82.61	(73.33, 92.34)	1.2434	(1.1884, 1.2936)	0.8619	(0.8237, 0.8966)
3e-2	49.10	(44.48, 54.24)	1.1466	(1.0978, 1.1949)	0.7948	(0.7609, 0.8282)
1e-2	28.46	(26.73, 30.33)	1.0652	(1.0199, 1.1068)	0.7383	(0.7070, 0.7672)
3e-3	14.20	(13.47, 14.98)	1.0385	(0.9956, 1.0806)	0.7199	(0.6901, 0.7490)
1e-3	6.46	(6.08, 6.84)	1.0293	(0.9909, 1.0664)	0.7134	(0.6868, 0.7392)
3e-4	2.18	(2.04, 2.33)	1.0220	(0.9831, 1.0604)	0.7084	(0.6815, 0.7350)
1e-4	0.75	(0.70, 0.80)	1.0220	(0.9823, 1.0610)	0.7084	(0.6809, 0.7355)
3e-5	0.35	(0.33, 0.39)	1.0200	(0.9781, 1.0633)	0.7070	(0.6779, 0.7370)
1e-5	0.34	(0.30, 0.38)	1.0025	(0.9461, 1.0550)	0.6949	(0.6558, 0.7313)
			$p_{\mathsf{llama1B}} \circ \mathsf{f}_\alpha$			
1e-1	129.22	(112.82, 146.19)	0.9922	(0.9467, 1.0405)	0.6877	(0.6562, 0.7212)
3e-2	90.74	(82.63, 99.33)	0.9128	(0.8682, 0.9542)	0.6327	(0.6018, 0.6614)
1e-2	47.81	(44.04, 51.52)	0.8626	(0.8252, 0.9027)	0.5979	(0.5720, 0.6257)
3e-3	26.15	(24.71, 27.73)	0.8402	(0.8037, 0.8790)	0.5823	(0.5571, 0.6093)
1e-3	14.89	(14.18, 15.69)	0.8364	(0.8001, 0.8725)	0.5797	(0.5546, 0.6048)
3e-4	6.80	(6.46, 7.18)	0.8358	(0.7999, 0.8724)	0.5793	(0.5545, 0.6047)
1e-4	2.62	(2.49, 2.77)	0.8362	(0.8018, 0.8699)	0.5796	(0.5558, 0.6030)
3e-5	1.42	(1.35, 1.49)	0.8361	(0.8002, 0.8717)	0.5796	(0.5547, 0.6042)
1e-5	0.73	(0.69, 0.77)	0.8360	(0.8002, 0.8717)	0.5795	(0.5546, 0.6042)

Table A.14: Cross-entropy in nats per byte and bits per byte, together with throughput, for various thresholds τ using $p_{\mathcal{X}} \circ \mathsf{f}_\alpha$, for p_{llama8B} and p_{phi4}, continuing Table A.13. 95% confidence intervals are given in parentheses. All configurations use 10 paragraphs.

τ	bytes / sec		Bits / byte		Cross-entropy	
	Mean	95% CI	Mean	95% CI	Mean	95% CI
			$p_{\mathsf{llama8B}} \circ \mathsf{f}_\alpha$			
1e-1	88.65	(79.18, 99.61)	0.8176	(0.7757, 0.8661)	0.5667	(0.5377, 0.6003)
3e-2	76.39	(71.27, 81.35)	0.7392	(0.7007, 0.7759)	0.5124	(0.4857, 0.5378)
1e-2	52.05	(49.41, 54.93)	0.7153	(0.6778, 0.7507)	0.4958	(0.4698, 0.5203)
3e-3	28.81	(27.18, 30.44)	0.6959	(0.6627, 0.7282)	0.4824	(0.4594, 0.5047)
1e-3	16.86	(16.01, 17.82)	0.6913	(0.6584, 0.7251)	0.4791	(0.4564, 0.5026)
3e-4	9.73	(9.24, 10.25)	0.6870	(0.6513, 0.7179)	0.4762	(0.4515, 0.4976)
1e-4	4.35	(4.15, 4.58)	0.6870	(0.6557, 0.7221)	0.4762	(0.4545, 0.5005)
3e-5	2.60	(2.47, 2.73)	0.6869	(0.6533, 0.7208)	0.4761	(0.4528, 0.4996)
1e-5	1.57	(1.49, 1.66)	0.6869	(0.6552, 0.7180)	0.4761	(0.4541, 0.4977)
			$p_{\mathsf{phi4}} \circ \mathsf{f}_\alpha$			
1e-1	42.39	(37.95, 47.19)	0.8714	(0.8217, 0.9208)	0.6040	(0.5695, 0.6382)
3e-2	34.48	(31.73, 37.33)	0.7748	(0.7317, 0.8180)	0.5371	(0.5072, 0.5670)
1e-2	24.89	(23.05, 26.59)	0.7547	(0.7165, 0.7959)	0.5231	(0.4966, 0.5517)
3e-3	16.55	(15.34, 17.92)	0.7301	(0.6959, 0.7690)	0.5061	(0.4824, 0.5330)
1e-3	9.91	(9.41, 10.52)	0.7241	(0.6882, 0.7595)	0.5019	(0.4770, 0.5264)
3e-4	5.81	(5.51, 6.14)	0.7170	(0.6803, 0.7509)	0.4970	(0.4716, 0.5205)
1e-4	2.55	(2.41, 2.70)	0.7163	(0.6824, 0.7493)	0.4965	(0.4730, 0.5194)
3e-5	1.54	(1.46, 1.62)	0.7162	(0.6793, 0.7490)	0.4965	(0.4709, 0.5192)
1e-5	0.86	(0.81, 0.91)	0.7162	(0.6828, 0.7502)	0.4964	(0.4733, 0.5200)

A.3.5 Benchmarking the computational shortcut

We benchmark the computational shortcut inherent in the prefix decomposition described in Section 3.2, which allows us to decompose the precover into remainder and quotient representatives. We generate suboptimal decompositions by randomly selecting n IP-universal states in the transducer (see Section 3.2) and treating them as non-IP-universal in our algorithms (see Section 4.2). This gradually decreases the size of the quotient and correspondingly increases the size of the remainder. Table A.15 shows the average JSD between the original and modified distributions over the first 256 bytes

of each of the first five paragraphs of the `wikitext-2-raw-v1` dataset (Merity et al., 2017). For each value of n, we repeat this random selection three times and report the mean JSD.

The distributions diverge rapidly, and after we convert roughly 15–20% of IP-universal states, the algorithm repeatedly encounters dead ends. The exact fraction varies between runs. As shown in Figure A.3, each repeat exhibits a distinct step function: JSD remains low until a particular high-connectivity state is converted, then jumps by 2–3 orders of magnitude and plateaus. In the PTB transducer, one IP-universal state (state 193) is a routing hub with 274 arcs spanning 245 output symbols. When this state is converted, some source prefixes that would have entered the quotient $\mathtt{Q}$ remain in the expansion loop, making pruning more likely to reach a dead end and trigger backtracking (Algorithm A.3). Different runs encounter this hub at different values of n depending on the permutation order, producing the staircase pattern. This shows that the effect of converting IP-universal states depends strongly on the states selected, with high-connectivity states producing the largest changes in the resulting distribution.

Table A.15: Average Jensen–Shannon divergence (JSD) over the first 256 bytes of each of the first five WikiText-2 paragraphs after randomly converting n IP-universal states of the PTB transducer to non-IP-universal, with τ =1e-3. Each value of n uses three random conversion orders. Parentheses give 95% confidence intervals, and JSD is computed against the unmodified transducer ($n = 0$).

Converted States (n)	$p_{\mathsf{llama1B}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$	$p_{\mathsf{llama8B}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$	$p_{\mathsf{phi4}} \circ \mathtt{f}_{\alpha} \circ \mathtt{f}_{\mathrm{ptb}}$
0	(not applicable)	(not applicable)	(not applicable)
2	3.9e-5 (3.5e-5, 4.3e-5)	1.3e-4 (7.4e-5, 2.1e-4)	6.6e-5 (3.1e-5, 1.1e-4)
5	9.5e-4 (7.5e-4, 1.2e-3)	9.0e-4 (7.5e-4, 1.1e-3)	7.8e-4 (6.4e-4, 9.4e-4)
8	9.8e-4 (8.0e-4, 1.2e-3)	9.2e-4 (7.7e-4, 1.1e-3)	8.1e-4 (6.6e-4, 9.6e-4)
11	3.0e-2 (2.6e-2, 3.3e-2)	2.8e-2 (2.5e-2, 3.1e-2)	2.6e-2 (2.3e-2, 2.9e-2)
14	2.9e-2 (2.6e-2, 3.3e-2)	2.8e-2 (2.5e-2, 3.1e-2)	2.6e-2 (2.3e-2, 2.9e-2)
16	2.9e-2 (2.6e-2, 3.2e-2)	2.8e-2 (2.5e-2, 3.2e-2)	2.6e-2 (2.3e-2, 2.9e-2)
19	2.9e-2 (2.6e-2, 3.3e-2)	2.8e-2 (2.5e-2, 3.2e-2)	2.6e-2 (2.3e-2, 2.9e-2)

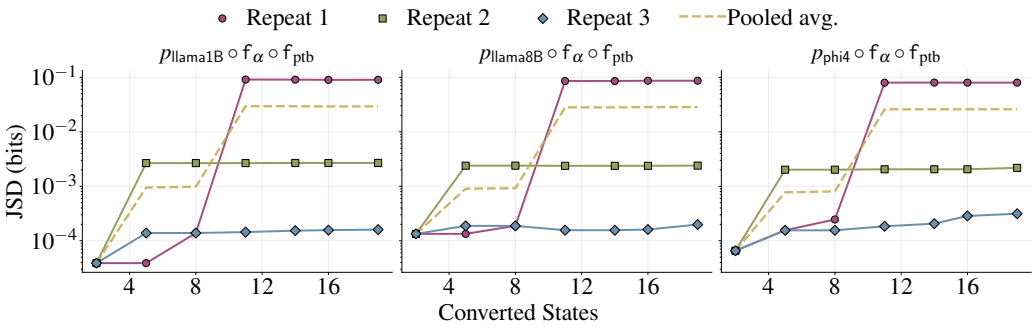


Figure A.3: Per-repeat Jensen–Shannon divergence (JSD) vs. number of states converted from IP-universal to non-IP-universal for the PTB transducer (τ =1e-3). Each repeat uses a different random conversion order. The abrupt jumps occur when individual high-connectivity states are converted.

Appendix B

Stochastic estimation: algorithms, proofs, and results

This appendix contains material reproduced and adapted from Snæbjarnarson et al. (2026b), with editorial changes for narrative consistency.

B.1 Sequential Monte Carlo over the precover

This section starts by developing the with-replacement SMC algorithms for estimating the total mass of the precover, including the generic estimator and its resampling (Section B.1.1), the intermediate targets and the `smc_rb` algorithm (Section B.1.2), and the unbiasedness of `smc_rb` (Section B.1.3). It then gives the common tail-roulette prune for the fixed-M methods (Section B.1.4) and the proof of `beam_swor_adaptive` (Section B.1.5).

B.1.1 Sequential Monte Carlo

The sum we wish to estimate (Equation (3.2)) is over source strings filtered by a membership test in the precover $\mathcal{P}(\boldsymbol{y})$. A **potential** is a function $\phi\colon \mathcal{X}^* \to \mathbb{R}_{\geq 0}$ that weights each source string (Doucet, Freitas, and Gordon, 2001), and Z is its expectation under the source model,

$$Z \stackrel{\text{def}}{=} \sum_{\boldsymbol{x} \in \mathcal{X}^*} p_{\mathcal{X}}(\boldsymbol{x})\phi(\boldsymbol{x})\,. \tag{B.1}$$

For a transduced model, we take the potential to be the predicate `is_member` (Chapter 4), $\phi(\boldsymbol{x}) = \texttt{is_member}(\boldsymbol{x}, \boldsymbol{y})$, for a fixed $\boldsymbol{y}$. The estimand Z is then

```
def smc_simple(q→, φ, M):
  R ← ∅
  P ← ([ε, 1/M])_{m=1}^{M}
  while |P| > 0:
    P' ← ∅
    for (x, w) ∈ P:
      sample x' ~ q→(· | x)
      w ← w · p→_X(x' | x) / q→(x' | x)
      if x' = EOS:
        R.add((x, w · φ(x)))
      else:
        P'.add((x x', w))
    P ← resample(P')
  return Σ_{(x,w)∈R} w

def resample(P):
  if |P| = 0: return ∅
  if Σ_i w^(i) = 0: return ∅
  if ESS(P) ≥ η |P|:
    return P
  j_1, ..., j_{|P|} ~ i.i.d. Categorical(w^(i) / Σ_i w^(i))
  return {(x^(j_k), Σ_i w^(i) / |P|)}_{k=1}^{|P|}
```

Algorithm B.1: A minimal **sequential Monte Carlo** algorithm for estimating Z (Equation (B.1)). M particles extend one source symbol at a time under the proposal and are reweighted according to Equation (B.2). Completed particles receive the potential factor. When the ESS falls below its threshold, `resample` samples with replacement and resets the particle weights to their mean (Equation (B.5)).

the probability that a string drawn from the source model maps to a target string having $\boldsymbol{y}$ as a prefix, i.e., $Z = \overrightarrow{p_Y}(\boldsymbol{y})$.

The algorithm estimating Z with SMC is shown in Algorithm B.1, and we now describe it. The pool $\mathcal{P}$ is initialized with M **particles** at the empty string, each with weight $1/M$ (Chapter B). At every iteration, a particle draws its next symbol by **sequential importance sampling** (Doucet, Freitas, and Gordon, 2001), drawing from a **conditional proposal** distribution $\overrightarrow{q}(\cdot \mid \boldsymbol{x})$ (Chapter B) and correcting for it by multiplying its weight by the ratio of source to proposal conditionals (Chapter B),

$$w \leftarrow w \cdot \frac{\overrightarrow{p_X}(x' \mid \boldsymbol{x})}{\overrightarrow{q}(x' \mid \boldsymbol{x})}. \tag{B.2}$$

A particle that draws EOS is **completed**: it leaves the pool, and its weight, multiplied by the potential, is added to the estimate $\hat{Z}$ (Chapter B). Assume for a moment that we do not call `resample`. Then, for a single particle, the

factors of Equation (B.2), the potential, and the initial $1/M$ multiply to $\frac{1}{M}w(\boldsymbol{x})$, with the **importance weight**

$$w(\boldsymbol{x}) \stackrel{\text{def}}{=} \phi(\boldsymbol{x}) \frac{\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})}{\overrightarrow{q}(\text{EOS} \mid \boldsymbol{x})} \prod_{s=1}^{|\boldsymbol{x}|} \frac{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}_s \mid \boldsymbol{x}_{<s})}{\overrightarrow{q}(\boldsymbol{x}_s \mid \boldsymbol{x}_{<s})} = \frac{\phi(\boldsymbol{x}) p_{\mathcal{X}}(\boldsymbol{x})}{q(\boldsymbol{x})}. \tag{B.3}$$

Here $q(\boldsymbol{x}) \stackrel{\text{def}}{=} \overrightarrow{q}(\text{EOS} \mid \boldsymbol{x}) \prod_{s=1}^{|\boldsymbol{x}|} \overrightarrow{q}(\boldsymbol{x}_s \mid \boldsymbol{x}_{<s})$. We further require that

$$\phi(\boldsymbol{x})\, p_{\mathcal{X}}(\boldsymbol{x}) > 0 \implies q(\boldsymbol{x}) > 0 \qquad \text{for every } \boldsymbol{x} \in \mathcal{X}^*, \tag{B.4}$$

to ensure that every string with positive potential–weighted mass can be sampled.

Without the `resample` call the particles are independent and the estimate is the average $\hat{Z} = \frac{1}{M}\sum_{m=1}^{M} w(\boldsymbol{x}^{(m)})$ of M draws of Equation (B.3).

Without resampling, the particle weights can become increasingly unequal as source prefixes grow, until a single particle dominates (Doucet, Freitas, and Gordon, 2001). The **effective sample size**

$$\text{ESS} = \frac{\left(\sum_{m=1}^{|\mathscr{P}|} w^{(m)}\right)^2}{\sum_{m=1}^{|\mathscr{P}|} \left(w^{(m)}\right)^2} \tag{B.5}$$

quantifies that imbalance. When it falls below $\eta|\mathscr{P}|$ (Chapter B), for a fixed $0 < \eta < 1$, `resample` redraws the pool with replacement proportionally to weight and resets every weight to the pool mean (Chapter B), so the higher-weighted particles are duplicated and the lower ones discarded. When the support condition Equation (B.4) holds and the run halts almost surely, the estimator is unbiased without resampling. Since multinomial resampling with replacement preserves weighted sums in conditional expectation, `smc_simple` remains unbiased.

B.1.2 Intermediate targets and the `smc_rb` algorithm

In Section B.1.1, particles are extended until EOS is sampled and only then is the potential evaluated. `beam_summing` instead tests prefixes as they are extended and drops a prefix once it can no longer cover the target. `smc_rb` applies SMC within `beam_summing`'s target recursion. Like `beam_summing`, `smc_rb` applies the `is_cylinder` and `is_member` checks to add quotient and remainder contributions. It samples one live next source symbol per particle instead of enumerating all of them. The recursion iterates over target prefixes

while each call extends source prefixes. To express these updates as importance sampling, associate each target position t with the **intermediate target**

$$\mu_t(\boldsymbol{x}) \stackrel{\text{def}}{=} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \cdot \psi_t(\boldsymbol{x}). \tag{B.6}$$

As in Section 5.1.1, we rely on the live twist $\psi_t(\boldsymbol{x}, \boldsymbol{y}) \stackrel{\text{def}}{=} \texttt{is_live}(\boldsymbol{x}, \boldsymbol{y}_{\leq t})$, written $\psi_t(\boldsymbol{x})$ when $\boldsymbol{y}$ is clear. Sequential importance sampling requires each intermediate target to place mass only where the preceding target does (Del Moral, Doucet, and Jasra, 2006; Naesseth, Lindsten, and Schön, 2019). This holds because a prefix that can still cover $\boldsymbol{y}_{\leq t}$ can also cover $\boldsymbol{y}_{<t}$.

Recall the live set $\mathcal{A}(\boldsymbol{x})$ of next source symbols (Section 5.1.1), taken here relative to $\boldsymbol{y}_{\leq t}$, and define the **twisted live mass**

$$\lambda_t(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{x'' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x'' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x''). \tag{B.7}$$

The proposal distribution draws the next symbol with probability proportional to $\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x')$,

$$\begin{aligned}
\overrightarrow{q_t}(x' \mid \boldsymbol{x}) &= \frac{\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x')}{\nu_t(\boldsymbol{x})}, \\
\overrightarrow{q_t}(\text{EOS} \mid \boldsymbol{x}) &= \frac{\mathbb{1}\{\texttt{is_member}(\boldsymbol{x}, \boldsymbol{y}_{\leq t})\}\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})}{\nu_t(\boldsymbol{x})}, \\
\nu_t(\boldsymbol{x}) &\stackrel{\text{def}}{=} \mathbb{1}\{\texttt{is_member}(\boldsymbol{x}, \boldsymbol{y}_{\leq t})\}\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) + \sum_{x'' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x'' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x''),
\end{aligned} \tag{B.8}$$

For the live twist, $\lambda_t(\boldsymbol{x})$ reduces to the raw conditional probability of the live set, so every particle stays live for $\boldsymbol{y}_{\leq t}$ by construction. `smc_simple` uses the same live-set restriction. The downside is that the live set carries no information about how much covering mass lies ahead, so a particle may look good early and carry little weight later (Figure B.1b). We accept the compromise because the live predicate is computable at every step (Section 4.3.2), whereas the full mass ahead is the optimal twist of Equation (5.3), exactly the quantity we cannot compute.

Importance sampling in the target recursion. `smc_rb` iterates over the target and source indices as shown in the example Figure B.1a. A **target step** occurs once when the recursion advances from target position $t-1$ to t. At a fixed source prefix, the weight becomes

$$w \;\leftarrow\; w \cdot \frac{\mu_t(\boldsymbol{x})}{\mu_{t-1}(\boldsymbol{x})} = w \cdot \frac{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\psi_t(\boldsymbol{x})}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\psi_{t-1}(\boldsymbol{x})} = w \cdot \frac{\psi_t(\boldsymbol{x})}{\psi_{t-1}(\boldsymbol{x})}. \tag{B.9}$$

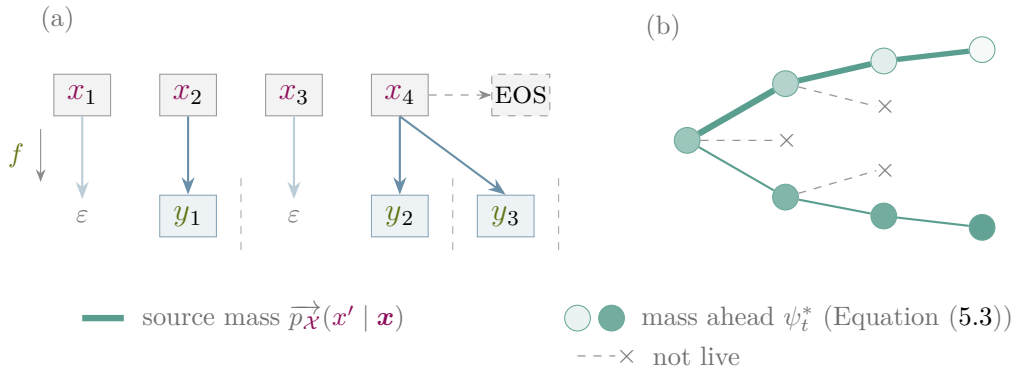


Figure B.1: (a) An example source string aligned to a target prefix under a function f. Each source symbol emits zero or more target symbols, and the dashed lines mark target-position boundaries. (b) The live twist excludes dead branches (×) but does not account for the covering mass beyond each live child. A high-probability branch with little mass ahead (top) may therefore be favored over a lower-probability branch with more mass ahead (bottom). Importance weighting corrects for the proposal, while resampling with replacement mitigates particle-weight degeneracy (Sections B.1.1 and B.1.3).

The **source step** advances by drawing one symbol from the twisted proposal. Its incremental importance weight is

$$\frac{\mu_t(\boldsymbol{x}x')/\mu_t(\boldsymbol{x})}{\overrightarrow{q_t}(x' \mid \boldsymbol{x})} = \frac{\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\psi_t(\boldsymbol{x}x')}{\psi_t(\boldsymbol{x})} \cdot \frac{\nu_t(\boldsymbol{x})}{\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\psi_t(\boldsymbol{x}x')} = \frac{\nu_t(\boldsymbol{x})}{\psi_t(\boldsymbol{x})}. \tag{B.10}$$

Therefore,

$$x' \sim \overrightarrow{q_t}(\cdot \mid \boldsymbol{x}), \qquad (\boldsymbol{x}, w) \leftarrow \left(\boldsymbol{x}x', w\frac{\nu_t(\boldsymbol{x})}{\psi_t(\boldsymbol{x})}\right), \tag{B.11}$$

with $\nu_t(\boldsymbol{x})$ the proposal's normalizer of Equation (B.8). At position t, the target-step factor is combined with one source-step factor for every symbol sampled within the call. A particle that ends position $t-1$ at the prefix $\boldsymbol{x} = \boldsymbol{x}_{\le n}$, $n = |\boldsymbol{x}|$, and takes k source steps through position t extends it to $\boldsymbol{x}_{\le n+k}$, drawing the symbol $\boldsymbol{x}_{n+j}$ at step j, and updates its weight by

$$w \leftarrow w \cdot \frac{\mu_t(\boldsymbol{x}_{\le n+k})}{\mu_{t-1}(\boldsymbol{x}_{\le n}) \prod_{j=1}^{k} \overrightarrow{q_t}(\boldsymbol{x}_{n+j} \mid \boldsymbol{x}_{<n+j})} = w \cdot \frac{\psi_t(\boldsymbol{x}_{\le n})}{\psi_{t-1}(\boldsymbol{x}_{\le n})} \prod_{j=1}^{k} \frac{\nu_t(\boldsymbol{x}_{<n+j})}{\psi_t(\boldsymbol{x}_{<n+j})}, \tag{B.12}$$

the target step once and one source-step factor per symbol.

For the live twist, $\psi_t(\boldsymbol{x}) = 1$ on every surviving particle. `smc_rb` marginalizes the EOS decision and conditions the proposal of Equation (B.8) on

```
def smc_rb(y, M, ψ, prune):
  N ← |y|
  if N = 0:
    (Q′, R′), Ẑ ← ([(ε, 1/M)]_{m=1}^{M}, ∅), [ ]
  else:
    (Q′, R′), Ẑ ← smc_rb(y_{<N}, M, ψ, prune)
  P ← QUEUE(Q′)
  (Q, R), A ← (∅, ∅), 0
  for (x, w) ∈ R′:
    if is_member(x, y):
      R.add((x, w))
      A += w
  while |P| > 0:
    P′ ← ∅
    for x, w ∈ P:
      if is_cylinder(x, y):
        Q.add((x, w))
        A += w
        continue
      if is_member(x, y):
        R.add((x, w · p→_X(EOS | x)))
        A += w · p→_X(EOS | x)
      λ ← Σ_{x′∈X} p→_X(x′ | x) · ψ(xx′)
      if λ = 0:
        continue
      sample x′ ~ p→_X(· | x) ψ(x·)/λ
      P′.add((xx′, (w·λ)/ψ(xx′)))
    P ← prune(P′, A)
  Ẑ.append(A)
  return (Q, R), Ẑ

def resample_{ψ,η}(P′):
  W^(i) ← w^(i) · ψ(x^(i))
  V ← Σ_i W^(i)
  N ← |P′|
  if V² ≥ η N Σ_i (W^(i))²:
    return P′
  j_1, ..., j_N ~^{i.i.d.} Categorical(W^(i)/V)
  return {(x^(j_n), (V w^(j_n))/(N W^(j_n)))}_{n=1}^{N}
```

Algorithm B.2: The smc_rb algorithm and ESS-gated multinomial resampling.

drawing a live next source symbol. Substituting the conditional proposal into Equation (B.2), the update simplifies to

$$w^{(m)} \leftarrow w^{(m)} \cdot \lambda_t(\boldsymbol{x}^{(m)}). \tag{B.13}$$

The sampled particle's weight therefore equals the sum of the weights of its parent's live children.

For a quotient particle carried from position $t-1$, the target-step factor $\psi_t(\boldsymbol{x})/\psi_{t-1}(\boldsymbol{x})$ is one if the prefix remains live at the new position and zero otherwise. It re-enters the pool unchanged when the factor is one. When the factor is zero, $\lambda_t(\boldsymbol{x}) = 0$, so smc_rb drops it at its next source step. The recursive call also returns completed remainder entries. The membership re-check retains one exactly when its source string covers the longer target. Together,

these operations implement the update from μ_{t-1} to μ_t (Algorithm B.2).

B.1.3 Unbiasedness of `smc_rb`

We now show that `smc_rb` is unbiased under the live twist. The proof follows Section 5.1.3 for `beam_summing` but with a single sampled symbol extension.

Proposition B.1 (Unbiasedness of `smc_rb`). *Fix a nonempty target string $\boldsymbol{y}$ and run `smc_rb` (Algorithm B.2) with the live twist $\psi = $ `is_live`. Suppose that every prune preserves the pool's expected weight at every prefix (Equation (5.12)) and that the run halts almost surely. Then $\mathbb{E}[\hat{Z}_{|\boldsymbol{y}|}] = \overrightarrow{p_Y}(\boldsymbol{y})$.*

Proof. We follow the nested induction used in Lemma 5.2, with an outer induction on target length and an inner induction on the within-call iterations. Write $\mathscr{P}_i$ for the `smc_rb` pool at iteration i and $\overline{\mathscr{P}}_i$ for the corresponding pool in the prune-free `beam_summing` run. In the outer base case, the M copies of ε with weight $1/M$ in `smc_rb` have the same aggregate weight as the single copy of weight one in `beam_summing`. The recursive initial-pool argument is otherwise unchanged, so it remains to replace the extension identity Equation (5.15).

Condition on the current pool and fix a non-cylinder particle at $\boldsymbol{x}$ with weight w. Write $\lambda(\boldsymbol{x}) = \sum_{x''} \overrightarrow{p_X}(x'' \mid \boldsymbol{x})\psi(\boldsymbol{x}x'')$ for its twisted live mass. When $\lambda(\boldsymbol{x}) > 0$, let X be drawn from the proposal of Equation (B.8), conditioned on a live source symbol. For any next source symbol x' with $\psi(\boldsymbol{x}x') > 0$, its conditional expected contribution is

$$\mathbb{E}_X\left[\mathbb{1}\{X = x'\}\frac{w\,\lambda(\boldsymbol{x})}{\psi(\boldsymbol{x}x')}\right] = \frac{\overrightarrow{p_X}(x' \mid \boldsymbol{x})\psi(\boldsymbol{x}x')}{\lambda(\boldsymbol{x})}\frac{w\,\lambda(\boldsymbol{x})}{\psi(\boldsymbol{x}x')} = w\,\overrightarrow{p_X}(x' \mid \boldsymbol{x}). \tag{B.14}$$

A zero-twist child receives no contribution under either extension. If $\lambda(\boldsymbol{x}) = 0$, both extensions assign every admitted child zero weight, and cylinder prefixes produce no children under either extension. Summing Equation (B.14) over the particles at $\boldsymbol{x}$ gives the conditional-expectation version of Equation (5.15). The tower rule and Equation (5.12) then close the same within-call induction and give, for every iteration i and source prefix $\boldsymbol{x}$,

$$\mathbb{E}[w_{\mathscr{P}_i}(\boldsymbol{x})] = w_{\overline{\mathscr{P}}_i}(\boldsymbol{x}). \tag{B.15}$$

Marginalizing the stop decision replaces its sampled contribution by

its conditional expectation. At a member, the baseline proposal stops with probability $\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})/\nu_t(\boldsymbol{x})$ and returns weight $w\nu_t(\boldsymbol{x})$, so

$$\frac{\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})}{\nu_t(\boldsymbol{x})} w\, \nu_t(\boldsymbol{x}) = w\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}), \tag{B.16}$$

which is the remainder contribution added by `smc_rb`. `smc_rb` adds quotient contributions and re-queues prefixes exactly as `beam_summing` does. The same assembly argument as in Theorem 5.1 therefore applies to the pool identity above, and Equations (5.17a, Equation (5.2) and Tonelli) and (5.17d, Equation (3.6)) give the result. ■

The ESS-gated multinomial resample of Algorithm B.2 satisfies Equation (5.12) for every fixed input pool (Chopin and Papaspiliopoulos, 2020, Prop. 16.3).

B.1.4 A common tail-roulette prune

Source-prefix enumeration can continue indefinitely even as the remaining particle weights tend to zero (Section 5.1.4). The decorator in Algorithm B.3 independently removes each sufficiently light particle with probability 1/2 and doubles its weight otherwise. For use with this decorator, we represent each particle as a triple $(\boldsymbol{x}, w, w_0)$. At the start of a target-prefix call, we set $w_0 = w$ for every particle in $\mathscr{P}_1$. Each child and resampled copy inherits w_0. The original pruning function acts on the source prefix and current weight while keeping w_0 unchanged. For $\kappa \in (0, 1)$, the decorator first applies the *original* pruning function, then applies the coin flip independently to each returned particle $(\boldsymbol{x}, w, w_0)$ with $w < \kappa w_0$.

Fix a finite starting pool $\mathscr{P}_1$, $\kappa \in (0, 1)$, and a pruning function that returns at most M particles. We first show that the decorator preserves the particle weight in expectation when the original pruning function does. We then show that the algorithms with a particle cap M in Table 5.1 halt almost surely.

Expected-weight preservation. Conditional on the original pruning function's output and the reference weights w_0, a particle selected for the coin flip contributes $2w$ with probability 1/2 and zero otherwise, giving expected contribution w. The coin flip therefore preserves every prefix weight in conditional expectation. Averaging over the original pruning function's output, the decorated pruning function satisfies Equation (5.12) whenever the original does.

```
def with_tail_roulette(prune, κ):
  def decorated(𝒫′, ...):
    𝒫 ← prune(𝒫′, ...)
    𝒫̃ ← ∅
    for (x^(m), w^(m), w_0^(m)) ∈ 𝒫:
      if w^(m) = 0:
        continue
      if w^(m) ≥ κ w_0^(m):
        𝒫̃.add((x^(m), w^(m), w_0^(m)))
        continue
      b ∼ Bernoulli(1/2)
      if b = 1:
        𝒫̃.add((x^(m), 2w^(m), w_0^(m)))
    return 𝒫̃
  return decorated
```

Algorithm B.3: Tail-roulette decorator for a pruning function.

Almost-sure halting. Let $\mathscr{P}_i^\circ$ be the pool returned by the original pruning function at iteration i, immediately before the coin flips, and define its total particle weight by

$$W_i \stackrel{\text{def}}{=} \sum_{(\boldsymbol{x}^{(m)}, w^{(m)}, w_0^{(m)}) \in \mathscr{P}_i^\circ} w^{(m)}. \tag{B.17}$$

By Equation (5.12) and Lemma 5.2, $\mathbb{E}[W_i]$ equals the total weight in the corresponding prune-free pool after i source extensions for `beam_summing` and hence `beam_swor`. The sampled-extension identity Equation (B.14) gives the same result for `smc_rb`, and the importance-weight identity Equations (B.2) and (B.3) gives it for `smc_simple`. For $\texttt{prune}_M$, the expected total is no larger because its prune only removes particles. In every case,

$$\mathbb{E}[W_i] \le \sum_{m=1}^{|\mathscr{P}_1|} w_0^{(m)} \sum_{\boldsymbol{x}' \in \mathcal{X}^i} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x}^{(m)}) \xrightarrow{i \to \infty} 0. \tag{B.18}$$

The inner sum is the conditional probability of generating at least i additional source symbols from $\boldsymbol{x}^{(m)}$. Because $p_{\mathcal{X}}$ is a distribution over finite strings, this probability tends to zero as $i \to \infty$ (Section 2.2). Since the outer sum has finitely many terms, the displayed limit follows.

Discard zero-weight particles from $\mathscr{P}_1$. If none remain, the call halts immediately. Otherwise, set

$$\delta_{\min} \stackrel{\text{def}}{=} \kappa \min_{1 \le m \le |\mathscr{P}_1|} w_0^{(m)} > 0. \tag{B.19}$$

Because descendants inherit w_0, every descendant's coin-flip threshold κw_0 is at least $\delta_{\min}$.

For $i \geq 1$, write $\widetilde{\mathscr{P}}_i$ for the pool after the coin flips at iteration i. If $W_i < \delta_{\min}$, every particle in $\mathscr{P}_i^{\circ}$ is selected for a coin flip. The original pruning function returns at most M particles, so the independent coin flips empty the pool with conditional probability at least 2^{-M}. Since W_i is nonnegative, Markov's inequality and the coin-flip bound above give

$$\Pr[W_i \geq \delta_{\min}] \leq \frac{\mathbb{E}[W_i]}{\delta_{\min}} \longrightarrow 0, \tag{B.20}$$

$$\Pr\left[\widetilde{\mathscr{P}}_i \neq \emptyset\right] \leq \Pr[W_i \geq \delta_{\min}] + (1 - 2^{-M}) \Pr\left[\widetilde{\mathscr{P}}_{i-1} \neq \emptyset\right], \qquad i \geq 2. \tag{B.21}$$

Let $E_i \stackrel{\text{def}}{=} \{\widetilde{\mathscr{P}}_i \neq \emptyset\}$ be the event that the pool remains nonempty after iteration i, and let $p_i \stackrel{\text{def}}{=} \Pr[E_i]$. Because an empty pool remains empty, $E_{i+1} \subseteq E_i$. Thus, $(p_i)_{i \geq 1}$ is nonincreasing and bounded below by zero, so it converges to some $p \geq 0$. Taking $i \to \infty$ in the recursion, with $p_i \to p$, $p_{i-1} \to p$, and $\Pr[W_i \geq \delta_{\min}] \to 0$, gives $p \leq (1 - 2^{-M})p$. Since $1 - 2^{-M} < 1$, this forces $p = 0$. The intersection $\bigcap_{i \geq 1} E_i$ is the event that the pool remains nonempty at every iteration. For decreasing events, its probability is the limit of their probabilities, so $\Pr\left[\bigcap_{i \geq 1} E_i\right] = \lim_{i \to \infty} p_i = 0$. A call using the decorated prune thus halts almost surely. For the recursive algorithms, the target string has finitely many positions, so the complete run also halts almost surely. Any $\kappa \in (0, 1)$ gives the halting guarantee.

Settings. The threshold κ determines when roulette begins. We choose it small so that roulette is delayed until a particle's current weight is small relative to its reference weight. We use $\kappa = e^{-15}$ for the WikiText experiments and $\kappa = e^{-30}$ for the DNA and MECO experiments. `beam_swor_adaptive` does not require the decorator because its adaptive prune uses $A_{\epsilon,t}$ to ensure almost-sure halting (Proposition 5.1).

B.1.5 Proof for adaptive-budget SWOR

By Theorem 5.1, it is enough to show that every call to the adaptive-budget prune preserves the expected weight at each source prefix (Equation (5.12)) and that the full run halts almost surely.

Proposition 5.1 (Adaptive-budget SWOR). *Fix a positive integer M, a twist ψ, $\rho \in (0, 1]$, and $\epsilon > 0$, and run* `beam_swor_adaptive` *using $A_{\epsilon,t}$*

from Equation (5.20). *Then (I) each prune call satisfies Equation* (5.12). *Under the live twist* $\psi = \texttt{is_live}$, *(II) the full run halts almost surely. Consequently, under the live twist,* $\mathbb{E}[\hat{Z}_t] = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}_{\leq t})$ *at every position.*

Proof. (I) To establish Equation (5.12), consider a single prune call with $\mathscr{P}'$ and $A_{\epsilon,t}$ fixed. For particle i, write $\mathcal{W}^{(i)} = w^{(i)}\psi(\boldsymbol{x}^{(i)})$ and $\mathcal{I}_+ = \{i : \mathcal{W}^{(i)} > 0\}$, as in Algorithm 5.2. These inputs determine W, μ, and the rounded value m before the prune draws any randomness. So they determine which of the three cases below applies, even though the roulette case may subsequently change m from zero to one. We consider the following three cases.

(1) Suppose $W = 0$. The extension step ensures $\psi(\boldsymbol{x}^{(i)}) > 0$ for every particle in $\mathscr{P}'$ (Section 5.1.1). Since $W = \sum_i \mathcal{W}^{(i)} = 0$ and $\mathcal{W}^{(i)} \geq 0$, every $\mathcal{W}^{(i)} = 0$. Because $\psi(\boldsymbol{x}^{(i)}) > 0$, this implies $w^{(i)} = 0$ for every particle. Hence $w_{\mathscr{P}'}(\boldsymbol{x}) = w_{\emptyset}(\boldsymbol{x}) = 0$ for every $\boldsymbol{x}$, which establishes Equation (5.12).

(2) Suppose $W > 0$ and $m \geq 1$. We split according to whether $|\mathcal{I}_+| \leq m$ or $|\mathcal{I}_+| > m$. If $|\mathcal{I}_+| \leq m$, the prune keeps all particles in $\mathcal{I}_+$ unchanged. For every $i \notin \mathcal{I}_+$, $\mathcal{W}^{(i)} = 0$. Since $\psi(\boldsymbol{x}^{(i)}) > 0$, this implies $w^{(i)} = 0$. Removing those particles therefore leaves the weight at every source prefix unchanged, so Equation (5.12) holds. If $|\mathcal{I}_+| > m$, the prune must select exactly m particles. It computes their inclusion probabilities from the twisted weights with `pps`, draws the particles, and applies Horvitz–Thompson reweighting. Here $\pi_i > 0$ for every $i \in \mathcal{I}_+$. Taking $h^{(i)} = \mathbb{1}\{\boldsymbol{x}^{(i)} = \boldsymbol{x}\}$ in Equation (5.9) gives

$$\mathbb{E}\left[w_{\widetilde{\mathscr{P}}}(\boldsymbol{x})\right] = w_{\mathscr{P}'}(\boldsymbol{x}),$$

which is Equation (5.12).

(3) Suppose $W > 0$ and $m = 0$. Because $W > 0$, the definition of μ in Equation (5.19) gives $\mu > 0$. Because $m = 0$, the rounding rule in the same equation gives $\mu < \frac{1}{2}$. Thus the division by μ below is well defined. Returning the empty pool deterministically would discard positive weight and violate Equation (5.12), so the prune instead uses this same μ as its roulette survival probability. It draws $B \sim \text{Bernoulli}(\mu)$ and returns the empty pool if $B = 0$. If $B = 1$, it sets $m = 1$ and replaces every $w^{(i)}$ by $w^{(i)}/\mu$ before performing a one-particle SWOR prune. The resulting twisted weights are $\mathcal{W}^{(i)}/\mu$. This common rescaling leaves $\mathcal{I}_+$ unchanged and, if `pps` is called, leaves its inclusion probabilities π_i unchanged. The $m \geq 1$ case above

therefore gives

$$\mathbb{E}\left[w_{\widetilde{\mathscr{P}}}(\boldsymbol{x}) \mid B = 1\right] = \frac{w_{\mathscr{P}'}(\boldsymbol{x})}{\mu}.$$

Averaging over B gives

$$\mathbb{E}\left[w_{\widetilde{\mathscr{P}}}(\boldsymbol{x})\right] = \mu \frac{w_{\mathscr{P}'}(\boldsymbol{x})}{\mu} + (1-\mu)0 = w_{\mathscr{P}'}(\boldsymbol{x}).$$

Thus Equation (5.12) also holds in the third case.
Together, the three cases establish (I). Multiplying Equation (5.12) by $\psi(\boldsymbol{x})$ and summing over source prefixes gives

$$\mathbb{E}\left[\sum_{\boldsymbol{x}} w_{\widetilde{\mathscr{P}}}(\boldsymbol{x})\psi(\boldsymbol{x})\right] = \sum_{\boldsymbol{x}} w_{\mathscr{P}'}(\boldsymbol{x})\psi(\boldsymbol{x}) = W.$$

This is the form needed in (II).

(II) We prove by induction on target-prefix length that every call made by `beam_swor_adaptive` halts almost surely. The empty-prefix call halts after one iteration. Fix a nonempty target prefix $\boldsymbol{y}$, let $t = |\boldsymbol{y}|$, and suppose the recursive call for $\boldsymbol{y}_{<t}$ has returned. Its quotient is a finite starting pool, which we write as

$$\mathscr{P}_1 = \{(\boldsymbol{x}^{(j)}, w^{(j)})\}_{j=1}^{J}.$$

It is enough to prove halting conditional on this pool and on the preceding estimate $\hat{Z}_{t-1}$, so we hold both fixed below.

If $\mathscr{P}_1$ has no positive weight, the call halts immediately when the pool is empty or no later than its first prune by case (1) of (I). We may therefore assume that $\mathscr{P}_1$ contains a positive weight. When $t = 1$, $\hat{Z}_0 = 1$. When $t > 1$, every positive entry in $\mathscr{P}_1$ was added to the quotient of the preceding call, and its weight was included in $\hat{Z}_{t-1}$. Hence $\hat{Z}_{t-1} > 0$ and, throughout the current call,

$$A_{\epsilon,t} \geq \epsilon \hat{Z}_{t-1} > 0. \tag{B.22}$$

For $n \geq 1$, when the n^{th} prune is reached, let $\mathscr{P}'_n$ be the child pool passed to it and define $W_n \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}} w_{\mathscr{P}'_n}(\boldsymbol{x})\psi(\boldsymbol{x})$. If the current target-prefix call halts before this prune, define $W_n = 0$. Let μ_n and m_n denote the capped count and its rounded value before roulette at the n^{th} prune, defining both to be zero if that prune is not reached or if $W_n = 0$.

Part (I) preserves the expected weight at every source prefix through each adaptive prune. By (I) and iterated expectation, a length-n descendant $\boldsymbol{x}^{(j)}\boldsymbol{x}'$ has expected weight at most $w^{(j)}\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x}^{(j)})$. Summing over the live descendants gives

$$\mathbb{E}[W_n] \leq \sum_{j=1}^{J} w^{(j)} \sum_{\boldsymbol{x}' \in \mathcal{X}^n} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x}^{(j)}) \xrightarrow{n\to\infty} 0.$$

The inner sum is the conditional probability of generating at least n additional source symbols from $\boldsymbol{x}^{(j)}$. It tends to zero because $p_{\mathcal{X}}$ is a distribution over finite strings (Section 2.2), and the outer sum has finitely many terms.

At the n^{th} prune, write $\mathcal{W}_n^{(i)}$ for its twisted particle weights, so $W_n = \sum_i \mathcal{W}_n^{(i)}$. Then Equations (B.22) and (5.19) and $\min(1, x) \leq x$ give

$$\begin{aligned}\mu_n = \sum_i \min\left(1, \frac{M\mathcal{W}_n^{(i)}}{\rho\,(A_{\epsilon,t} + W_n)}\right) &\leq \frac{M}{\rho\,(A_{\epsilon,t} + W_n)} \sum_i \mathcal{W}_n^{(i)} \\ &= \frac{MW_n}{\rho\,(A_{\epsilon,t} + W_n)} \leq \frac{MW_n}{\rho\,\epsilon\hat{Z}_{t-1}}.\end{aligned}$$

If this prune is not reached, the last inequality still holds because $\mu_n = W_n = 0$ by definition. Taking expectations therefore gives

$$\mathbb{E}[\mu_n] \leq \frac{M}{\rho\,\epsilon\hat{Z}_{t-1}} \mathbb{E}[W_n] \xrightarrow{n\to\infty} 0.$$

The rounding rule in Equation (5.19) gives $m_n > 0$ exactly when $\mu_n \geq \frac{1}{2}$. Markov's inequality now gives

$$\Pr[m_n > 0] = \Pr\left[\mu_n \geq \tfrac{1}{2}\right] \leq 2\mathbb{E}[\mu_n] \xrightarrow{n\to\infty} 0. \tag{B.23}$$

Let E_n be the event that the call survives the n^{th} prune. Write $p_n \stackrel{\text{def}}{=} \Pr[E_n]$ and $p_0 = 1$. The events E_n are decreasing, so p_n converges to some $p \geq 0$. If $m_n = 0$ and $W_n > 0$, roulette lets the pool survive with probability $\mu_n < \frac{1}{2}$. If $W_n = 0$, the prune returns the empty pool. Conditioning on the state before the prune therefore gives

$$p_n \leq \Pr[m_n > 0] + \tfrac{1}{2}p_{n-1}.$$

Taking $n \to \infty$ and using Equation (B.23) gives $p \leq \frac{1}{2}p$, so $p = 0$. The events E_n are decreasing, and their intersection is the event that the call never halts. Therefore

$$\Pr\left[\bigcap_{n\geq 1} E_n\right] = \lim_{n\to\infty} p_n = 0.$$

Thus the current call halts almost surely for every realized finite starting pool. The induction proves that the run halts almost surely. Applying Theorem 5.1 to each target prefix using (I) and (II) gives the final claim. ■

B.2 Sensitivity to pruning parameters

The experiments use $\epsilon = 0$. For each fixed target prefix, the finite-context source models and finite-state transducers give finitely many live configurations. Full support gives a positive lower bound on the probability of reaching a member or cylinder from any such configuration, so A becomes positive almost surely. The proof of Proposition 5.1 then applies with $A_{\epsilon,t} = A > 0$. Besides M, the adaptive-budget prune uses ρ (Section 5.1.4), and the ESS-gated multinomial prune uses η (Section B.1). We vary both on the ten-paragraph bigram corpus of Section 5.2.1, using eight seeds (Figure B.2).

Smaller ρ retains more particles. At nominal M=8000, reducing ρ from 1 to 0.02 lowers the standard deviation from 0.711 to 0.073 nats and raises the time per seed from 349 to 1551 seconds. The default ρ=0.1 gives a standard deviation of 0.200 nats at 623 seconds. Plain `beam_swor` at M=2000 gives 0.170 nats at 3432 seconds. With 149 seeds, the standard deviation at the default is 0.163 with CI $[0.146, 0.184]$, consistent with the fixed-M `beam_swor` result while using about one fifth as much CPU time per seed. At M=2000, the between-seed standard deviation rises from 0.632 nats at ρ=0.1 to 4.815 at ρ=0.5 and 5.283 at ρ=1. At ρ=1, the mean live count is 18, and most log estimates are several nats low.

Increasing η causes more frequent resampling, while CPU time changes little across the tested values. At M=2000, the survivor-conditioned standard deviation rises from 5.1 nats at η=0.25, where seven of eight corpus seeds survive, to 7.86 at the default η=0.5 and 9.8 at η=0.75. At η=0, every seed has at least one zero paragraph estimate. The same holds at M=200 for every tested value of η, despite some nonzero individual paragraph estimates.

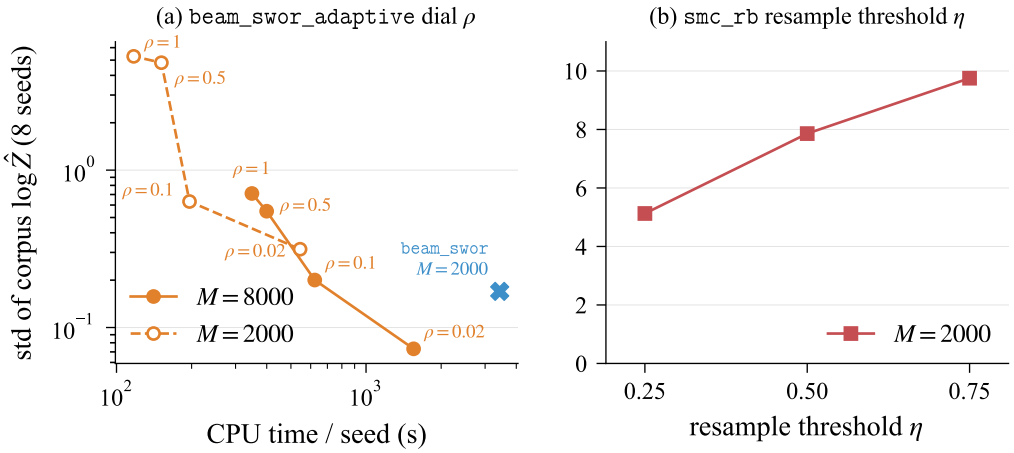


Figure B.2: Sensitivity to the pruning parameters on the ten-paragraph bigram corpus (eight seeds). (a) Between-seed standard deviation of the corpus $\log \hat{Z}$ against CPU time per seed as ρ varies for `beam_swor_adaptive` at nominal $M \in \{2000, 8000\}$. Labels give ρ, and plain `beam_swor` at $M=2000$ is the fixed-M reference. (b) Survivor-conditioned standard deviation as the ESS threshold η varies for `smc_rb` at $M=2000$. A corpus seed survives when all ten paragraph estimates are nonzero. Annotations mark settings with fewer surviving seeds and note that none survives at $M=200$.

B.3 Exact ground truth for bigram models

The bigram model can be represented as a weighted finite-state acceptor. Composing it with f_{ptb} and projecting onto the target labels gives a weighted automaton defining $p_{\text{bigram}} \circ \mathsf{f}_{\text{ptb}}$. In the representation used here, the functional transducer f_{ptb} admits at most one accepting path for each source string, so path sums do not double-count source strings.

For each state s, let $\beta(s)$ be its backward weight, the sum of the weights of all paths starting at s, including the final weight at the end of each path. Let T be the transition-weight matrix and f the final-weight vector. From each state, a path either stops there with its final weight or takes a transition and then continues. The backward weights therefore satisfy (Mohri, 2009)

$$\beta = f + T\beta, \qquad \text{so} \qquad \beta = (I + T + T^2 + \cdots) f = (I - T)^{-1} f. \tag{B.24}$$

Because the composed automaton has no cycle that can be traversed without consuming a source symbol, the number of consumed source symbols tends to infinity with the number of transitions. The probability of generating at least k source symbols tends to zero as $k \to \infty$, so the series converges.

For each position t and state s, let $\alpha_t(s)$ be the sum of the weights of all paths that start at the initial state, emit $\boldsymbol{y}_{\leq t}$, and end in s immediately after emitting $\boldsymbol{y}_t$. Summing over the states reached after emitting $\boldsymbol{y}_t$ gives

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}_{\leq t}) = \sum_{\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y}_{\leq t})} p_{\mathcal{X}}(\boldsymbol{x}) = \sum_{s \in \mathsf{S}} \alpha_t(s)\beta(s) = \alpha_t^\top \beta. \tag{B.25}$$

The backward weights $\beta(s)$ are computed once for all states, while $\alpha_t(s)$ is updated after each target symbol. The composed state space is finite for an n-gram source, but not for a neural language model whose next-symbol probabilities depend on the complete source prefix.

B.4 The genetic-code worst case

This appendix analyzes a worst case for the DNA setting in Section 5.2.3. Standard beam search captures an exponentially vanishing share of the target mass, while `smc_simple`, `smc_rb`, and fixed-M `beam_swor` are unbiased.

Example B.1 — Beam summing versus SMC on a uniform block code

Consider a uniform source model $p_{\mathcal{X}}(\boldsymbol{x}) = (1/4)^{3T}$ over source strings of length $3T$ on a four-symbol alphabet, and a target of length T where each target symbol is the image of d length-three source blocks ("codons"). Assume that the live proposal samples each of the d codons with probability $1/d$ and that the codons share their first two bases. This holds, for example, for repeated isoleucine, where $d = 3$. Exactly d^T strings in the source model's support belong to the precover, so $p_{\mathcal{Y}}(\boldsymbol{y}) = d^T \cdot (1/4)^{3T}$.

Standard beam search. `beam_summing` with $\texttt{prune}_M$ retains at most M of these d^T source strings, yielding

$$\hat{Z}_{\text{beam}} \leq M \cdot \left(\frac{1}{4}\right)^{3T}, \qquad \frac{\hat{Z}_{\text{beam}}}{p_{\mathcal{Y}}(\boldsymbol{y})} \leq \frac{M}{d^T}. \tag{B.26}$$

This ratio vanishes exponentially in T. For $T = 100$ and $d = 3$, the smallest beam that recovers all the mass has $M = 3^{100} \approx 5 \times 10^{47}$. The estimate is biased for any $M < d^T$, systematically underestimating $p_{\mathcal{Y}}(\boldsymbol{y})$.

Importance-weighted estimators. Each valid codon has source probability 4^{-3} and proposal probability $1/d$, so its importance weight gains

the factor $(1/4^3)/(1/d) = d/4^3$ under both `smc_simple` and `smc_rb` (Equations (B.2) and (B.13)). After T amino acids every particle carries

$$w^{(m)} = \frac{1}{M}\frac{d^T}{4^{3T}} = \frac{p_{\mathcal{Y}}(\boldsymbol{y})}{M}, \tag{B.27}$$

so $\hat{Z} = \sum_{m=1}^{M} w^{(m)} = p_{\mathcal{Y}}(\boldsymbol{y})$ exactly. The factor d^T emerges in the importance weights, letting M particles represent d^T covering source strings. Thus, without tail roulette, `smc_simple` and `smc_rb` both have zero variance, and fixed-M `beam_swor` is also exact. At every branch its child weights are equal, and its without-replacement reweighting preserves their total on every draw.

This is an extreme case, a uniform source and a constant alias count d. In practice, non-uniform DNA models introduce variance across particles. The example does not distinguish among the stochastic estimators, whose variance differences are measured in the non-uniform experiments.

B.5 Additional byte-to-word results

Figures B.3 and B.4 extend the exact bigram validation of Figure 5.3 to every byte position in all ten paragraphs. Table B.1 reports the additional per-paragraph GPT-2 estimates and runtimes.

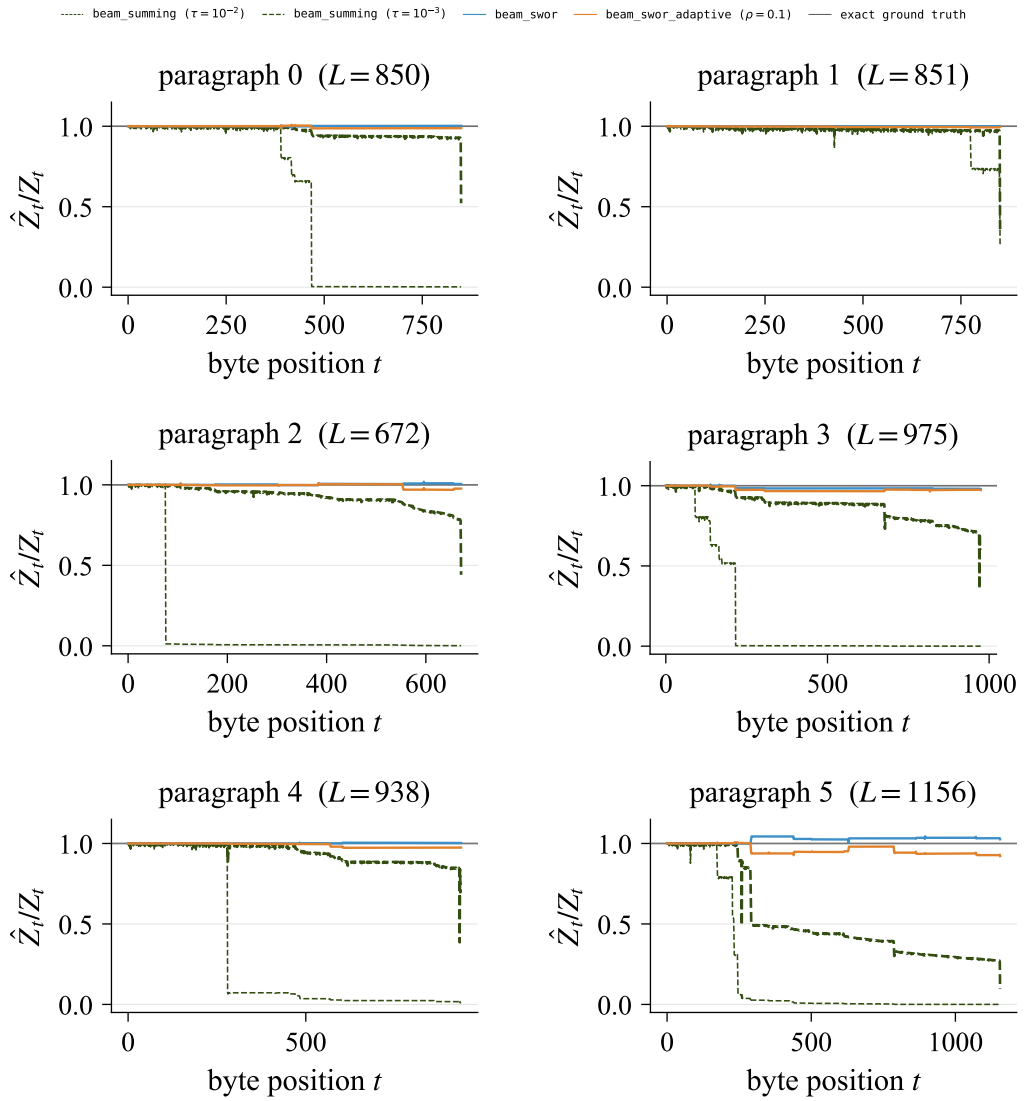


Figure B.3: Per-position ratio $\hat{Z}_t/Z_t$ for paragraphs 0–5 under $p_{\text{bigram}} \circ \mathsf{f}_{\text{ptb}}$ at M=200, averaged over 256 runs. Paragraphs 6–9 are shown in Figure B.4. The `beam_swor` and `beam_swor_adaptive` averages remain close to one with sampling variation. Threshold-pruned beam summing at τ=10^{-2} and τ=10^{-3} falls below one where pruning discards positive covering mass.

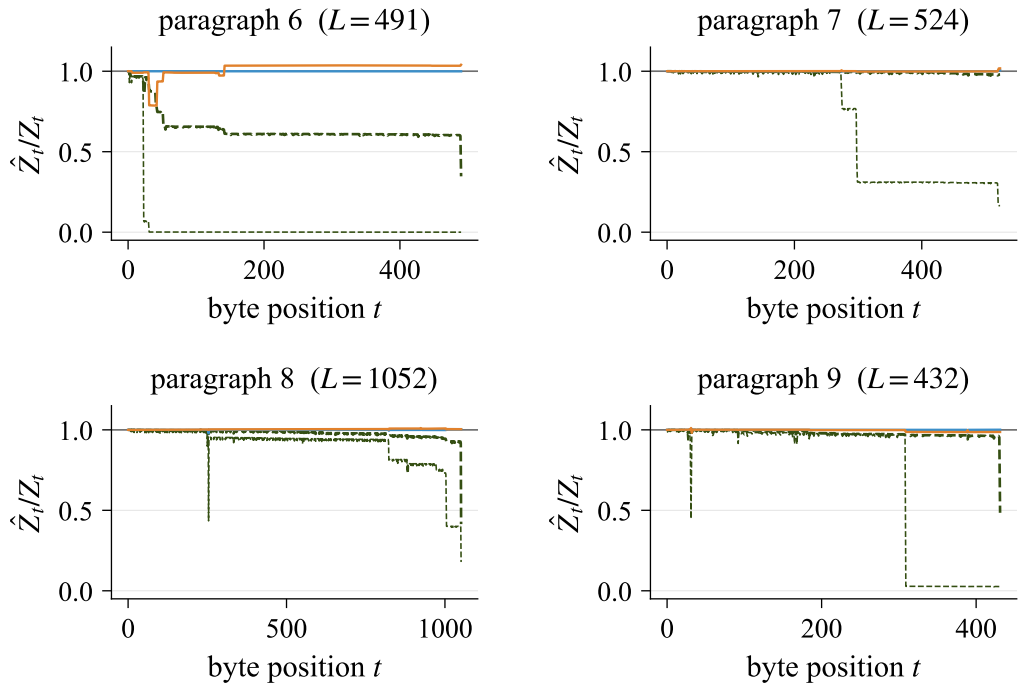


Figure B.4: Per-position ratio $\hat{Z}_t/Z_t$ for paragraphs 6–9 under $p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$ at M=200, averaged over 256 runs. Methods and axes match Figure B.3.

Table B.1: Per-paragraph log-prefix probabilities in nats and wall time in seconds under $p_{\mathsf{gpt2}} \circ \mathsf{f}_{\mathrm{ptb}}$. `beam_summing` uses $\tau = 10^{-3}$. `beam_swor_adaptive` uses $\rho = 0.1$, nominal $M = 800$, and thirty-two runs. Parentheses give the between-run standard deviation, and wall times are per run. The Δ column is `beam_summing` minus `beam_swor_adaptive`. The total row sums the per-paragraph log estimates. The corpus panel of Figure 5.4 instead averages the per-run corpus estimates before taking the logarithm, hence its -5376.7.

para	`beam_summing`	`beam_swor_adaptive`	Δ	t_{beam}	t_{adaptive}
0	-556.96	-554.96 (0.25)	-2.00	52	57
1	-564.10	-556.41 (1.38)	-7.70	106	210
2	-443.86	-443.21 (0.12)	-0.65	54	102
3	-747.96	-743.12 (0.19)	-4.84	170	216
4	-635.85	-631.16 (0.20)	-4.69	191	269
5	-772.31	-768.42 (0.27)	-3.88	353	647
6	-359.56	-355.23 (0.12)	-4.34	190	244
7	-330.81	-330.57 (0.13)	-0.24	21	21
8	-687.00	-684.43 (0.13)	-2.57	53	73
9	-311.62	-309.09 (0.27)	-2.54	33	39
total	-5410.04	-5376.60	-33.45	1224	1879

Appendix C

Surprisal: units, data, and regression details

This appendix contains material reproduced and adapted from Kiegeland et al. (2026), with editorial changes for narrative consistency.

This appendix supports Chapter 6 with details of the transformations behind the unit inventories, dataset processing, surprisal estimation, and the full regression specification and results.

C.1 Prefix-freeness of h

In Equation (6.2) we place SEP at the *right* edge of each unit, i.e., SEP marks a unit's completion, so that $h(u) \stackrel{\text{def}}{=} \boldsymbol{\xi}_u \,\text{SEP}$. One could equivalently consider the mirror convention

$$h_{\text{l}}(u) \stackrel{\text{def}}{=} \text{SEP}\, \boldsymbol{\xi}_u,$$

in which SEP instead marks a unit's *onset*. Both are monoid homomorphisms $\mathcal{U}^* \to \mathcal{Y}^*$ whose images $h(\mathcal{U})$ and $h_{\text{l}}(\mathcal{U})$ are regular and related by a 1-symbol shift of SEP. The pushforward identity Equation (6.3), however, is guaranteed under the completion convention. The reason, as we show below, is that the completion convention makes $h(\mathcal{U})$ a prefix-free code, whereas the onset convention does not guarantee prefix-freeness.

Reduction. Using $h(\boldsymbol{u}{\cdot}u) = h(\boldsymbol{u}){\cdot}h(u)$, Equation (6.3) reduces to the prefix identity

$$\rho(\boldsymbol{x}) \succeq \boldsymbol{u} \iff h(\rho(\boldsymbol{x})) \succeq h(\boldsymbol{u}), \tag{C.1}$$

for all $\boldsymbol{x} \in \mathcal{X}^*$ and $\boldsymbol{u} \in \mathcal{U}^*$. The ($\Rightarrow$) direction holds for any monoid homomorphism. Only the ($\Leftarrow$) direction depends on the placement of SEP.

Completion (sep trailing). Writing $\boldsymbol{u} = u_1 \cdot \cdots \cdot u_k$ and $\rho(\boldsymbol{x}) = u'_1 \cdot \cdots \cdot u'_T$, we have

$$h(\boldsymbol{u}) = \boldsymbol{\xi}_{u_1} \text{ SEP } \cdots \boldsymbol{\xi}_{u_k} \text{ SEP},$$
$$h(\rho(\boldsymbol{x})) = \boldsymbol{\xi}_{u'_1} \text{ SEP } \cdots \boldsymbol{\xi}_{u'_T} \text{ SEP}.$$

Since $\text{SEP} \notin \Xi$, every SEP in $h(\rho(\boldsymbol{x}))$ sits at a block boundary. Hence, if $h(\rho(\boldsymbol{x})) \succeq h(\boldsymbol{u})$, then the *final* SEP of $h(\boldsymbol{u})$ must coincide with the block boundary after $\boldsymbol{\xi}_{u'_k}$. Matching back block-by-block forces $u'_i = u_i$ for $i \leq k$, and thus $\rho(\boldsymbol{x}) \succeq \boldsymbol{u}$.

Onset (sep leading). Under h_{l}, the image $h_{\text{l}}(\boldsymbol{u})$ ends with $\boldsymbol{\xi}_{u_k}$ without a following SEP because SEP marks a unit's onset rather than its completion. A prefix match then admits any $u' \in \mathcal{U}$ whose underlying string begins with $\boldsymbol{\xi}_{u_k}$, not only u_k itself. Example 6.3 gives a concrete counterexample.

C.2 Unit transformations used in the experiments

The word-level transformations are implemented in Pynini (Gorman, 2016), a Python library for compiling and composing finite transducers that builds on OpenFST (Riley, Allauzen, and Jansche, 2009).

C.2.1 Characters

As shown in Chapter 4 (see Figure 4.1), the transformation from tokens to characters can be encoded using an FST. We do not compile the FST for the experiments in Section C.4. Instead, we use the algorithms and implementations of Vieira et al. (2025a) to transform $p_{\mathcal{X}}$ into a character-level model.

C.2.2 Acontextual words

Recent work argues that the common leading-whitespace convention of many BPE tokenizers, where the space preceding a word is bundled with the word's first token, misallocates surprisal, and that probability mass should instead be attributed as trailing whitespace, i.e., assigned to the preceding word (Oh and Schuler, 2024; Pimentel and Meister, 2024). At the same time, Giulianelli et al. (2024) argue that there is no universally correct convention and that the attribution of whitespace should be chosen to match the experimental setup. Under the definition of Oh and Schuler (2024) and Pimentel and Meister

(2024), leading-versus-trailing whitespace decoding can be interpreted as an aggregation method that specifies how probability mass is redistributed across unit boundaries. Here, we express leading and trailing attribution as two finite transducers, $\mathtt{f}_{\mathrm{L}}$ and $\mathtt{f}_{\mathrm{T}}$. See Example 6.4. This leading-versus-trailing choice concerns the *delimiter byte's* unit membership (i.e., which unit the space belongs to) and is a separate axis from the trailing-SEP convention on h itself. See Section C.1 for why we use the trailing-SEP convention to guarantee prefix-freeness. In our experiments, we set the delimiter set to $D = \{\textvisiblespace\}$, so whitespace is the sole signal of a unit boundary.

Note that a third variant, in which whitespace is absorbed into the separator ($q_1 \xrightarrow{\mathsf{d}:\mathrm{SEP}} q_0$) so that no unit ever contains a whitespace character, is easy to construct by replacing the delimiter arcs in Example 6.4.

C.2.3 Contextual words

The contextual-word transducer $\mathtt{f}_{\mathrm{ptb}}$ follows the construction in Section A.2.5. Each Penn Treebank tokenization rule is encoded as a small contextual string-rewrite transducer using Pynini's rewrite calculus (replace operations with explicit left/right contexts, Example 6.5), and the rule transducers are composed from left to right.

C.2.4 Transducer sizes

Table C.1 reports the number of states and arcs of each FST $\mathtt{f}$ used in the experiments, together with its number of *IP-universal* states, i.e., states for which the corresponding input projection accepts every source string $\boldsymbol{x} \in \mathcal{X}^*$ (see Section 4.3.3 for a detailed discussion). IP-universal states can be handled more efficiently by the algorithms of Chapter 4, so transducers with a larger IP-universal fraction offer more opportunities for this shortcut.

Transducer	States	Arcs	IP-universal
Acontextual (leading)	3	517	3 (all)
Acontextual (trailing)	258	1,029	258 (all)
Contextual	361	35,210	78 (partial)

Table C.1: Size of each finite transducer $\mathtt{f}$ used in the experiments, together with the number of IP-universal states (Section 4.3.3). "all" indicates that every state is IP-universal.

The two delimiter-insertion transducers $\mathtt{f}_{\mathrm{L}}$ and $\mathtt{f}_{\mathrm{T}}$ differ in size despite being drawn as equivalent three-state machines in Example 6.4. This is because the compiled FSTs allow only one output symbol per arc, whereas the figure uses a shorthand to draw arcs with multiple-symbol outputs. Each such arc is compiled into a chain of two arcs through an auxiliary state, with one auxiliary per input-byte value. In $\mathtt{f}_{\mathrm{L}}$ the multi-symbol output sits on the delimiter arc $q_1 \xrightarrow{\mathtt{d}:\mathrm{SEP}\mathtt{d}} q_0$, so only $|D|$ auxiliary states are introduced. With $D = \{\textvisiblespace\}$ minimization then merges q_B with q_0, whose outgoing arcs are identical, leaving three states. In $\mathtt{f}_{\mathrm{T}}$ the multi-symbol output sits on the arc $q_0 \xrightarrow{\mathtt{x}:\mathrm{SEP}\mathtt{x}} q_1$, so $|\mathcal{X} \setminus D|$ auxiliaries are introduced. With a byte alphabet ($|\mathcal{X}| = 256$) that is 255 auxiliaries whose distinct output labels prevent merging, giving $3 + 255 = 258$ states.

C.3 Dataset details

We reprocess the English portion of the MECO eye-tracking corpus (Siegelman et al., 2022), which contains scanpaths from 46 readers recorded while reading 12 short Wikipedia excerpts. Table C.2 reports the number of observations at each stage of the analysis pipeline, from raw units to the final GAMM input, for each of the unit inventories.

Inventory	n_{units}	n_{obs}	n_{lag}
GPT-2 tokens	2,478	41,553	40,589
Acontextual (leading)	2,095	40,254	39,290
Acontextual (trailing)	2,095	40,731	39,767
Contextual	2,264	34,478	33,514
Characters	13,226	54,578	53,614

Table C.2: Pipeline observation counts per unit inventory. n_{units}: total units across all 12 trials. n_{obs}: per-reader observations after excluding zero reading times (unfixated units). n_{lag}: after dropping the first two observations of each (reader, trial) pair, which lack values for the spillover lags. See Table C.7 for the final counts entering the GAMM.

	GPT-2	Acontextual	Contextual
GPT-2 tokens	—	70.8%	81.4%
Acontextual	70.9%	—	85.0%
Contextual	84.2%	87.8%	—

Table C.3: **Overlap with whitespace stripped**: percentage of units in the row inventory whose form, after removing any attributed whitespace, is also a unit in the column inventory. The acontextual leading and trailing variants yield identical stripped forms and are merged here.

	GPT-2	Acontextual (leading)	Acontextual (trailing)	Contextual
GPT-2 tokens	—	70.5%	0.0%	2.1%
Acontextual (leading)	71.0%	—	0.0%	0.7%
Acontextual (trailing)	0.0%	0.0%	—	0.3%
Contextual	2.2%	0.7%	0.3%	—

Table C.4: **Overlap with whitespace kept** (the unit text used by our GAMMs): percentage of units in the row inventory that appear *verbatim*, including any attributed whitespace, as units in the column inventory.

C.3.1 Unit overlap

With whitespace stripped (Table C.3), the three word-like inventories share 71–88% of their units pairwise. With whitespace kept (Table C.4), the two acontextual variants become nearly disjoint, and the contextual inventory, whose units never contain whitespace, is nearly disjoint from all three. GPT-2 tokens, which use leading-space by convention, overlap tightly with acontextual leading but not with acontextual trailing.

In the whitespace-stripped comparison, the 604 GPT-2 occurrences absent from the acontextual inventory are mainly punctuation marks that BPE splits off (126 commas, 96 periods) and BPE subword fragments (e.g., `Jan`, `us` from *Janus*). Conversely, the 325 acontextual occurrences absent from GPT-2 are words that BPE splits into multiple tokens (e.g., *Janus*, *thylacine*, *performance-enhancing*). Acontextual and contextual words differ mainly on punctuation attachment: the contextual inventory splits off 129 commas and 11 periods that the acontextual inventory attaches to the preceding word (e.g., acontextual ORGANIZATIONS, vs. contextual ORGANIZATIONS | ,). Possessives ('S) and quotation marks (“, ”) account for the remaining contextual-only units.

C.3.2 Unit and fixation visualizations

Figures C.1 to C.5 show the same trial (Reader 3, Text 1 from the MECO English corpus) segmented under the unit inventories used in our experiments, together with the recorded fixation data. Colored backgrounds mark unit boundaries, and each fixation is colored by the unit it is attributed to. In the contextual (PTB) inventory, a sentence-final period is typically split off as its own unit only when it is followed by EOS. Periods that end a sentence mid-trial remain attached to the preceding word.

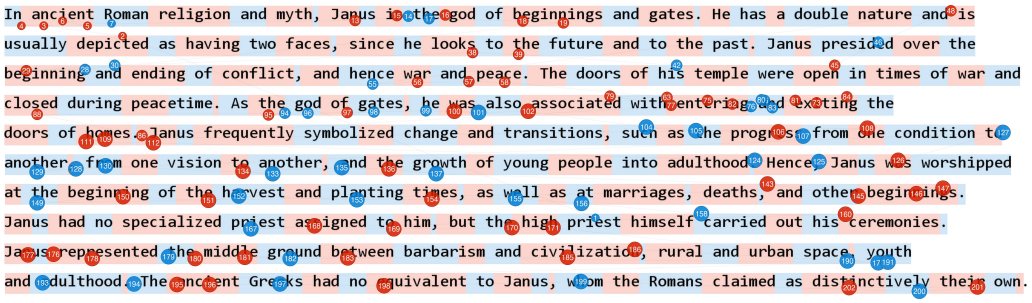


Figure C.1: Units and fixations for **model tokens** (BPE, leading delimiter). Reader 3, Text 1 (MECO English).

C.4 Experimental details

For the word inventories, we use the algorithms of Chapter 4 to evaluate GPT-2 Small[1] (Radford et al., 2019) from the 🤗 Hugging Face hub (Wolf

[1] openai-community/gpt2

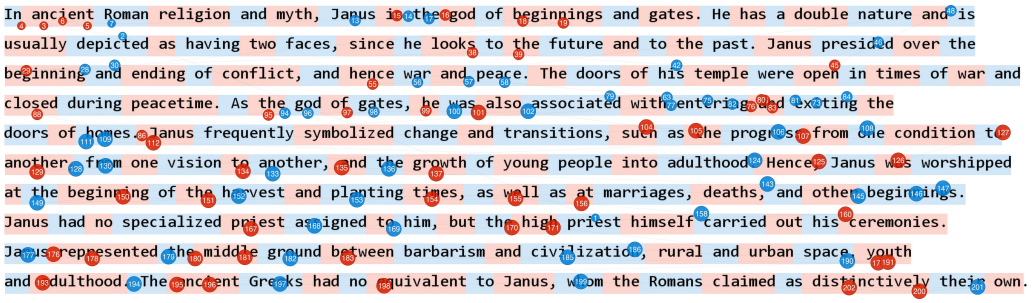


Figure C.2: Units and fixations for **acontextual (leading) words**. Reader 3, Text 1 (MECO English).

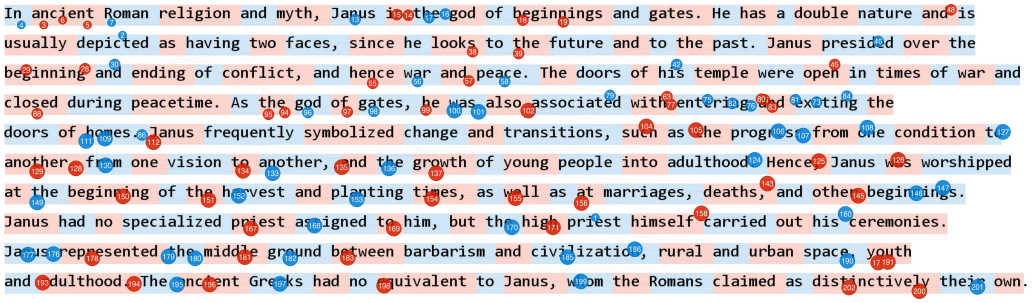


Figure C.3: Units and fixations for **acontextual (trailing) words**. Reader 3, Text 1 (MECO English).

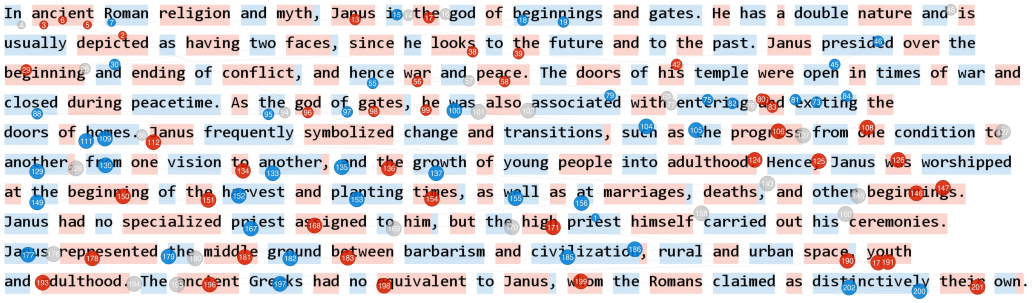


Figure C.4: Units and fixations for **contextual words**. Reader 3, Text 1 (MECO English).

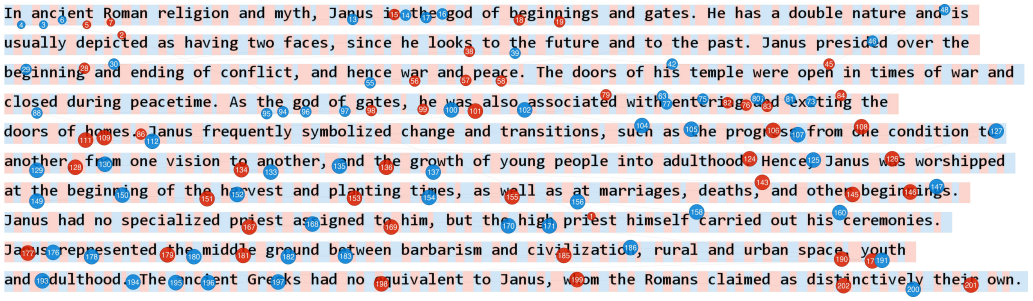


Figure C.5: Units and fixations for **character-level units**. Reader 3, Text 1 (MECO English).

et al., 2020) composed with the transducers described in Section C.2. To convert token-level models to character-level, we use **GenLM.bytes**.[2] We use vLLM to accelerate the underlying language-model calls (Kwon et al., 2023).

C.4.1 Computing surprisal

Both contextual surprisal and unigram surprisal are computed under the transduced language model $p_{\mathcal{Y}} = p_{\mathcal{X}} \circ \mathtt{f}$, from the next-unit conditional distribution $\overrightarrow{p_U}(u \mid \boldsymbol{u}) = \overrightarrow{p_Y}(h(u) \mid h(\boldsymbol{u}))$ of Equation (6.3). Following h as defined in Equation (6.2), each unit's byte extension ends with a SEP that marks its right boundary. Per-unit conditional probability is therefore the ratio

$$\overrightarrow{p_U}(u \mid \boldsymbol{u}_{<t}) \;=\; \frac{\overrightarrow{p_Y}\big(h(\boldsymbol{u}_{<t}) \cdot h(u)\big)}{\overrightarrow{p_Y}\big(h(\boldsymbol{u}_{<t})\big)}, \tag{C.2}$$

in which the numerator is a byte-level prefix probability closed off by the SEP carried in $h(u)$, making h prefix-free (see Footnote 9). The two quantities (contextual vs. unigram surprisal) differ only in how they consume this conditional: contextual surprisal scores the unit that actually occurred at each position, whereas unigram surprisal computes the marginal $\overrightarrow{p_U}(u) = \mathbb{E}_{\boldsymbol{u}_{<t}}[\overrightarrow{p_U}(u \mid \boldsymbol{u}_{<t})]$ with respect to contexts $\boldsymbol{u}_{<t}$ sampled from the LM, with u held fixed. Both share the hyperparameters listed in Table C.5.

Contextual surprisal. For the word inventories, we first apply the transducer $\mathtt{f}$ to each of the 12 MECO trials to obtain the transduced string $\boldsymbol{y} \in \mathcal{Y}^*$, then score its symbols left-to-right. Each step issues one call to the fast next-symbol decomposition (`decompose_next`) algorithm of Section 4.3.5, which returns the full next-symbol distribution over $\mathcal{Y}$. Consecutive calls are cached, so advancing the context from $\boldsymbol{y}_{<t}$ to $\boldsymbol{y}_{<t+1}$ extends the cached decomposition by a single symbol rather than recomputing from scratch. We prune low-probability beams during expansion using the thresholds in Table C.5.

Unigram surprisal. Unigram surprisal can be estimated by marginalizing the conditional probability of u over contexts from the same LM (Hopton et al., 2026). Because the full marginalization is intractable, we compute a Monte Carlo estimate by drawing S samples from the LM, summing the next-unit conditional $\overrightarrow{p_U}(u \mid \boldsymbol{u}_{<t})$ over all unit positions in those samples, and normalizing by $S + \sum_s |\boldsymbol{u}^{(s)}|$. For **GPT-2 tokens**, the conditional $\overrightarrow{p_U}(u \mid \boldsymbol{u}_{<t})$ is read directly from the LM. For **characters**, **GenLM.bytes** supplies byte-level conditionals, which are combined along each character's UTF-8 encoding.

[2] genlm-bytes

For **acontextual** and **contextual** inventories, units are defined through `f` and $\overrightarrow{p_{\mathcal{U}}}(u \mid \boldsymbol{u}_{<t})$ must be recovered from $\overrightarrow{p_{\mathcal{Y}}}$: we first transduce each sample from $p_{\mathcal{X}}$ to its target string $\boldsymbol{y}^s \in \mathcal{Y}^*$. We evaluate this conditional at each unit position, using the empty prefix as the context at the first position and the prefix through the preceding SEP at each subsequent position. For each such context $\boldsymbol{y}'$, we cache its log prefix probability $z = \log \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}')$ and score every candidate unit $u \in \mathcal{U}$ by evaluating $z_u = \log \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}' \cdot h(u))$, so that, following Equation (6.3), the next-unit conditional is $\overrightarrow{p_{\mathcal{U}}}(u \mid \boldsymbol{u}_{<t}) = \exp(z_u - z)$. The implementation uses pruning-based estimates of both prefix probabilities.

Efficient unigram scoring for the transduced LM. Rather than computing the full next-distribution (`decompose_next`), we use the single-symbol scoring routine of Chapter 4 (introduced for the cross-entropy evaluation of Section A.3.4), which decomposes only $\boldsymbol{y} \cdot y$ for a specified target symbol and returns $\log \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y} \cdot y)$ directly, skipping expansion over the full target alphabet and the final normalization. We apply this routine one target symbol at a time along the byte extension of each unit, computing successive log prefix probabilities and recovering the unit's conditional prefix probability by subtracting the cached log prefix probability at the boundary. Since all $|\mathcal{U}|$ unit extensions at a given boundary begin from the same prefix, their decomposition is shared across units, and units with common byte prefixes also reuse cached partial extensions.

Parameters. Table C.5 lists the hyperparameters for both computations. The beam-search and transduced-LM parameters are shared. The sampling block applies only to unigram estimation.

Parameter	Value	Role
GenLM.bytes (Vieira et al., 2025a)		
Beam size (K)	5	Maximum beam width: keeps the K highest-probability beams.
Beam prune threshold	0.001	Drop beams whose probability mass falls below this threshold.
Transduced-LM inference (Chapter 4)		
`f` prune threshold (τ)	0.005	Discard up to a fraction τ of the total candidate mass, starting with the least probable candidates.
Expansion lookahead	5	Stop expanding a bucket once all covering outputs extend this many symbols past the target and some quotient mass has been found.
PTB max expansion rounds	5	Halt PTB decomposition after this many expansion rounds.
Expand stop mass	0.01	Halt expansion once the relative remaining probability mass falls below this value.
Sampling for estimating unigram surprisal		
LM $p_{\mathcal{X}}$	GPT-2 Small	Source language model used in all experiments.
# samples S	500	Number of samples drawn from $p_{\mathcal{X}}$ in the unigram estimator.
Max length	50	Maximum number of tokens per sample.
Batch size	64	Batch size used when sampling from $p_{\mathcal{X}}$.

Table C.5: Hyperparameters used to estimate contextual and unigram surprisal. The beam-search parameters apply to both estimators. The sampling parameters apply only to the unigram estimator.

C.4.2 GPU usage & runtime

We ran all experiments on NVIDIA GeForce RTX 4090 and RTX 3090 GPUs, each with 24 GB of GPU memory. Table C.6 reports scoring throughput for contextual surprisal. The acontextual FSTs process approximately 200 target symbols per second. At this rate, scoring the 12 MECO English trials takes approximately one minute on a single GPU. The contextual FST is more than an order of magnitude slower because of its large number of states and arcs (see Table C.1). Its many non-IP-universal states also provide fewer opportunities for the IP-universality shortcut. See Section C.2.4 for a brief discussion and Section 4.3.3 for a detailed discussion of IP-universality.

Table C.6: Contextual surprisal scoring throughput per transducer on GPT-2 Small, aggregated over the 12 MECO English trials. "Symbols" is the total number of $\mathcal{Y}$-symbols scored across all trials, and "Syms/s" is the corresponding throughput. The character-level model is omitted because character surprisal is obtained via Vieira et al. (2025a) rather than through an FST composition.

Transducer	Symbols	Time (s)	Syms/s
Acontextual (leading)	15,309	72.0	212.8
Acontextual (trailing)	15,309	75.2	203.6
Contextual	13,407	1117.7	12.0

Unigram estimation is considerably more expensive than contextual surprisal because, at every SEP-boundary of every sample, the cached decomposition must be extended along every unit in $\mathcal{U}$. On GPT-2, scoring a typical 211-byte sample takes on the order of 20 seconds for acontextual (leading), 60 seconds for acontextual (trailing), and 8 minutes for the contextual transducer. The relative ordering mirrors Table C.6 because the same FST-decomposition step is the bottleneck in both pipelines. At 500 samples per unit inventory, this translates to roughly 3 hours (acontextual leading), 8 hours (acontextual trailing), and 2–3 days (contextual) of sequential single-GPU compute, so unigram estimation in practice requires parallelization. Since the outer sample loop runs independently across s, we chunk the 500-sample runs into independent jobs. Some chunks hit the per-job wall-clock limit before completing, so the final number of successfully scored samples per inventory is 498 (characters and GPT-2 tokens), 496 (acontextual leading), 497 (acontextual trailing), and 470 (contextual).

C.5 GAMM specification

We model log reading time as a generalized additive mixed model (Wood, 2017) fitted with `bam()` from `mgcv` in R, using fast restricted maximum likelihood (`method="fREML"`, `discrete=TRUE`). Each continuous predictor enters the log-mean as a cubic regression spline with up to six basis functions (`bs="cr"`, `k=6`); the model further includes a random intercept for participant and by-participant random slopes for every continuous predictor.

Baseline model ($\widetilde{\varphi}$).

```
log(rt) ~ s(length, bs="cr", k=6)
  + s(length_prev, bs="cr", k=6)
  + s(length_prev2, bs="cr", k=6)
  + s(unigram_surprisal, bs="cr", k=6)
  + s(unigram_surprisal_prev, bs="cr", k=6)
  + s(unigram_surprisal_prev2, bs="cr", k=6)
  + s(length, participant, bs="re")
  + s(length_prev, participant, bs="re")
  + s(length_prev2, participant, bs="re")
  + s(unigram_surprisal, participant,
      bs="re")
  + s(unigram_surprisal_prev, participant,
      bs="re")
  + s(unigram_surprisal_prev2, participant,
      bs="re")
  + s(participant, bs="re")
```

Target model (φ). The target model adds contextual surprisal and its two spillover lags to the baseline:

```
log(rt) ~ ... [baseline terms] ...
  + s(surprisal, bs="cr", k=6)
  + s(surprisal_prev, bs="cr", k=6)
  + s(surprisal_prev2, bs="cr", k=6)
  + s(surprisal, participant, bs="re")
  + s(surprisal_prev, participant, bs="re")
  + s(surprisal_prev2, participant, bs="re")
```

Here `s(x, bs="cr", k=6)` denotes a cubic regression spline with 6 basis functions; `length` is unit length in characters (whitespace-inclusive: for the acontextual and model-tokens inventories, a unit's length includes any leading or trailing whitespace attributed to it by the transducer, so a mid-sentence word is one character longer than its raw spelling); `unigram_surprisal` is unigram surprisal (see Section 6.3.1); `surprisal` is contextual surprisal from the language model; `_prev` and `_prev2` denote spillover from the first and second preceding unit, respectively. Each predictor enters as both a population-level smooth and a by-participant random slope `s(x, participant, bs="re")`; a random intercept for participants is included via `s(participant, bs="re")`. For the character inventory, unit length is constant (every unit is a single character), so the `length`, `length_prev`, and `length_prev2` smooths are omitted from both the baseline and target formulas; the remaining terms and the random-effects structure are unchanged.

Paired permutation test. Significance of Δ_{llh} is assessed by a one-sided paired permutation test on the per-observation held-out log-likelihood differences between the target and baseline models, with $B = 1000$ sign-flip permutations.

C.6 Details of the MECO reanalysis

This section contains material reproduced and adapted from Snæbjarnarson et al. (2026b), with editorial changes for narrative consistency.

We compare contextual surprisals computed by beam summing with $\texttt{prune}_\tau$ and `swor_adaptive` on the same 2,264 PTB units in the 12 MECO trials (individually read paragraphs). We use the model specification and evaluation protocol described in Sections C.5 and 6.4. We change only the pruning function used to compute contextual surprisal and rerun the fits on the same observations (n=33,472). The $\texttt{prune}_\tau$ values are the telescoping estimates reported in Section 6.4, computed with a threshold of 5×10^{-3}. For `swor_adaptive`, we use ρ=0.1 and nominal M=8000, and average each prefix-probability estimate over 10 runs before taking logarithms. The reported GAMM fits use the resulting surprisals without a jackknife correction. The analysis below shows that applying the correction changes no conclusion.

Jackknife check. Averaging the prefix-probability estimates preserves unbiasedness, but taking the logarithm of an average introduces bias. To estimate it, we recompute each unit's surprisal ten times, omitting one run each time. If s is one unit's surprisal from all ten runs and $s_{\not r}$ is its surprisal with run r omitted, the jackknife estimates the bias as $9\big(\frac{1}{10}\sum_{r=1}^{10} s_{\not r} - s\big)$, which we subtract from s.

The correction raises the corpus log probability by 1.55 nats, equivalently lowering its surprisal by the same amount. It changes the 106-nat uncorrected difference reported in Section 6.5 to approximately 107 nats. Refitting with the corrected surprisals moves every held-out Δ_{llh} by at most 0.015×10^{-3} nats and changes neither the results in Table 6.1 nor the between-estimator comparison below.

Between-estimator comparison. The two PTB-word analyses reported in Table 6.1 use the same observations, baseline predictors, model specification, and cross-validation folds, and differ only in how contextual surprisal is estimated. One-sided paired sign-flip tests over the 12 held-out trials show no evidence that the fit using either estimate predicts reading times better than the other ($p \geq 0.068$ in both directions for every measure).

C.7 Additional results

Table C.8 reports detailed GAMM results for each reading-time measure: mean per-observation held-out log-likelihood for the baseline (LL_{bl}) and target (LL_{tgt}) models, along with the improvement Δ_{llh} and 95% trial-level bootstrap CIs. Absolute log-likelihoods are not comparable across unit inventories because the number and granularity of observations differ. Table C.7 reports the number of observations entering the regression for each inventory, after the additional exclusions specific to the GAMM input.

Inventory	n_{GAMM}
GPT-2 tokens	40,589
Acontextual (leading)	39,290
Acontextual (trailing)	39,767
Contextual	33,472
Characters	48,834

Table C.7: Final number of observations entering the GAMM per unit inventory after the n_{lag} stage in Table C.2. The contextual inventory excludes 42 observations with missing unigram surprisal for `` and '' because their candidate-scoring paths are pruned at every sampled context under the settings in Table C.5. The character inventory excludes 4,780 zero-surprisal sub-token byte positions at which the byte distribution is deterministic under BPE.

Table C.8: GAMM results across reading-time measures. LL_{bl} and LL_{tgt} are the mean per-observation held-out log-likelihoods of the baseline and target models. Δ_{llh} is the target-minus-baseline improvement, reported in units of 10^{-3} nats with its 95% trial-level bootstrap CI in brackets. Significance via paired permutation test (see Section C.5): $^{*}\,p < 0.05$, $^{**}\,p < 0.01$.

	First fixation				*Gaze duration*				*Total reading time*			
Inventory	LL_{bl}	LL_{tgt}	Δ_{llh}	p	LL_{bl}	LL_{tgt}	Δ_{llh}	p	LL_{bl}	LL_{tgt}	Δ_{llh}	p
Characters	−0.4958	−0.4957	$0.11_{[-0.16,\ 0.40]}$	0.145	−0.5008	−0.5008	$0.09_{[-0.15,\ 0.35]}$	0.185	−0.5662	−0.5661	$0.10_{[-0.22,\ 0.45]}$	0.171
GPT-2 tokens	−0.4725	−0.4720	$0.55^{*}_{[-0.37,\ 1.46]}$	0.013	−0.5958	−0.5943	$1.52^{**}_{[0.09,\ 2.85]}$	<0.001	−0.7562	−0.7537	$2.56^{**}_{[0.07,\ 4.98]}$	<0.001
Acontextual (leading)	−0.4699	−0.4697	$0.28_{[-0.69,\ 1.16]}$	0.065	−0.6102	−0.6088	$1.41^{**}_{[-0.14,\ 2.81]}$	<0.001	−0.7681	−0.7651	$3.00^{**}_{[0.48,\ 5.34]}$	<0.001
Acontextual (trailing)	−0.4692	−0.4686	$0.63^{**}_{[-0.67,\ 1.76]}$	0.004	−0.6061	−0.6044	$1.68^{**}_{[-0.12,\ 3.17]}$	<0.001	−0.7669	−0.7640	$2.91^{**}_{[-0.46,\ 5.35]}$	<0.001
Contextual	−0.4745	−0.4737	$0.81^{**}_{[-0.47,\ 1.94]}$	0.003	−0.5984	−0.5962	$2.13^{**}_{[0.43,\ 3.73]}$	<0.001	−0.7409	−0.7376	$3.24^{**}_{[-0.16,\ 6.32]}$	<0.001

Figure C.6 shows the approximate F-statistics for all fixed-effect smooth

terms in the full-data GAMM fit, broken down by predictor group and reading-time measure. We observe two patterns: First, the current-unit surprisal smooth is significant for every unit inventory and every reading-time measure. Second, length and spillover predictors contribute primarily to the word-like inventories (tokens, acontextual, contextual) and to the later measures (gaze duration and total reading time). At the character level, the length smooth is absent because all units share the same length.

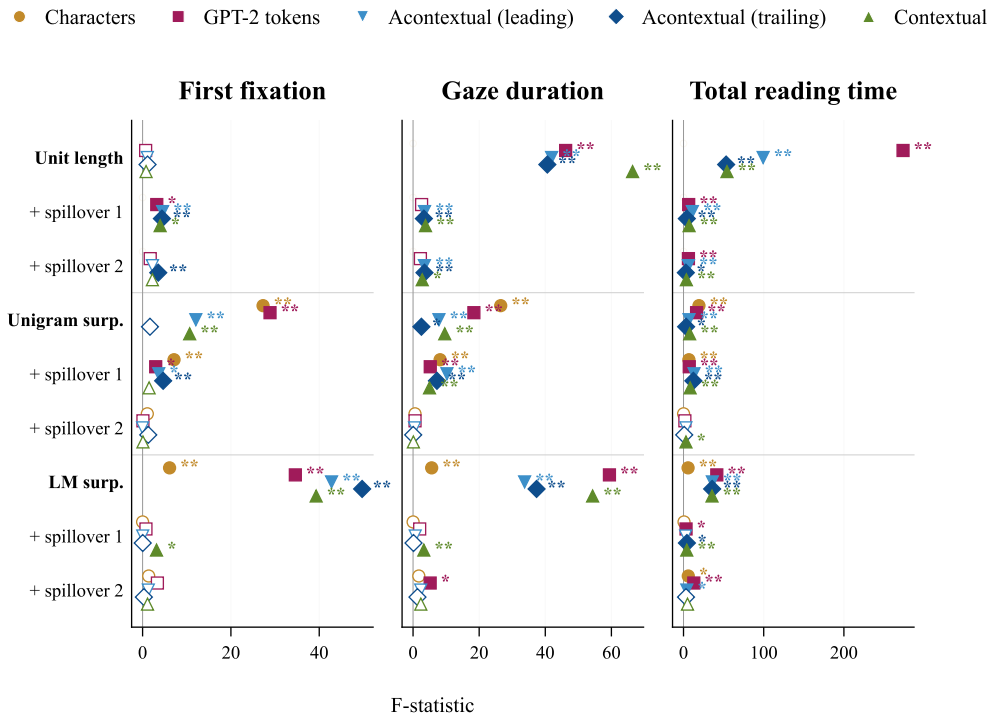


Figure C.6: Approximate F-statistics for fixed-effect smooth terms from the full-data GAMM fit, grouped by predictor type. Within each group, rows correspond to the current unit, spillover 1, and spillover 2. Filled markers indicate significance ($^{*}\,p < 0.05$; $^{**}\,p < 0.01$).

Bibliography

Abdin, Marah, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Russell J. Hewett, Mojan Javaheripi, Piero Kauffmann, James R. Lee, Yin Tat Lee, Yuanzhi Li, Weishung Liu, Caio C. T. Mendes, Anh Nguyen, Eric Price, Gustavo de Rosa, Olli Saarikivi, Adil Salim, Shital Shah, Xin Wang, Rachel Ward, Yue Wu, Dingli Yu, Cyril Zhang, and Yi Zhang (2024). *Phi-4 Technical Report*. arXiv: 2412.08905 [cs.CL]. URL: https://arxiv.org/abs/2412.08905.

Allauzen, Cyril, Bill Byrne, Adrià de Gispert, Gonzalo Iglesias, and Michael Riley (2014). "Pushdown Automata in Statistical Machine Translation". In: *Computational Linguistics* 40.3. URL: https://aclanthology.org/J14-3008/.

Arehalli, Suhas, Brian Dillon, and Tal Linzen (2022). "Syntactic Surprisal From Neural Models Predicts, But Underestimates, Human Processing Difficulty From Syntactic Ambiguities". In: *Proceedings of the Conference on Computational Natural Language Learning*. URL: https://aclanthology.org/2022.conll-1.20/.

Arús-Pous, Josep, Simon Viet Johansson, Oleksii Prykhodko, Esben Jannik Bjerrum, Christian Tyrchan, Jean-Louis Reymond, Hongming Chen, and Ola Engkvist (2019). "Randomized SMILES strings improve the quality of molecular generative models". In: *Journal of Cheminformatics* 11.71.

Athiwaratkun, Ben, Shiqi Wang, Mingyue Shang, Yuchen Tian, Zijian Wang, Sujan Kumar Gonugondla, Sanjay Krishna Gouda, Robert Kwiatkowski, Ramesh Nallapati, Parminder Bhatia, and Bing Xiang (2024). "Token Alignment via Character Matching for Subword Completion". In: *Findings of the Association for Computational Linguistics: ACL 2024*. URL: https://aclanthology.org/2024.findings-acl.929/.

Balota, David A., Alexander Pollatsek, and Keith Rayner (1985). "The interaction of contextual constraints and parafoveal visual information in reading". In: *Cognitive Psychology* 17.3. URL: https://doi.org/10.1016/0010-0285(85)90013-1.

Beesley, Kenneth R. and Lauri Karttunen (2003). *Finite State Morphology*. URL: https://press.uchicago.edu/ucp/books/book/distributed/F/bo3613750.html.

Beinborn, Lisa and Yuval Pinter (2023). "Analyzing Cognitive Plausibility of Subword Tokenization". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2023.emnlp-main.272/.

Berglund, Martin, Willeke Martens, and Brink van der Merwe (2024). "Constructing a BPE Tokenization DFA". In: *Implementation and Application of Automata*. ISBN: 978-3-031-71112-1. URL: https://link.springer.com/chapter/10.1007/978-3-031-71112-1_5.

Berglund, Martin and Brink van der Merwe (2023). "Formalizing BPE Tokenization". In: *Proceedings of the International Workshop on Non-Classical Models of Automata and Applications*. Vol. 388. URL: https://doi.org/10.4204/EPTCS.388.4.

Berstel, Jean (1979). *Transductions and Context-Free Languages*. Vol. 38. URL: https://www.worldcat.org/oclc/06364613.

Bjerrum, Esben Jannik (2017). "SMILES Enumeration as Data Augmentation for Neural Network Modeling of Molecules". In: *arXiv preprint arXiv:1703.07076*. URL: https://arxiv.org/abs/1703.07076.

Blanchard, Harry E., Alexander Pollatsek, and Keith Rayner (1989). "The acquisition of parafoveal word information in reading". In: *Perception & Psychophysics* 46.1. ISSN: 0031-5117. URL: https://link.springer.com/content/pdf/10.3758/BF03208078.pdf.

Bloomfield, Leonard (1933). *Language*. URL: https://archive.org/details/language0000leon_v8i3.

Bonchi, Filippo and Damien Pous (2015). "Hacking nondeterminism with induction and coinduction". In: *Communications of the ACM* 58.2. URL: https://doi.org/10.1145/2713167.

Brodbeck, Christian, Shohini Bhattasali, Aura AL Cruz Heredia, Philip Resnik, Jonathan Z Simon, and Ellen Lau (2022). "Parallel processing in speech perception with local and global representations of linguistic context". In: *eLife* 11. URL: https://doi.org/10.7554/eLife.72056.

Brothers, Trevor and Gina R. Kuperberg (2021). "Word predictability effects are linear, not logarithmic: Implications for probabilistic models of sentence comprehension". In: *Journal of Memory and Language* 116. URL: https://www.sciencedirect.com/science/article/pii/S0749596X20300887.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jef-

frey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei (2020). "Language Models are Few-Shot Learners". In: *Advances in Neural Information Processing Systems*. Vol. 33. URL: https://papers.nips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html.

Cao, Kris and Laura Rimell (2021). "You should evaluate your language model on marginal likelihood over tokenisations". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2021.emnlp-main.161/.

Chen, Charlie, Sebastian Borgeaud, Geoffrey Irving, Jean-Baptiste Lespiau, Laurent Sifre, and John Jumper (2023). "Accelerating Large Language Model Decoding with Speculative Sampling". In: *arXiv preprint arXiv:2302.01318*. URL: https://arxiv.org/abs/2302.01318.

Chirkova, Nadezhda, Germán Kruszewski, Jos Rozen, and Marc Dymetman (2023). "Should you marginalize over possible tokenizations?" In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. URL: https://aclanthology.org/2023.acl-short.1/.

Choffrut, Christian (1977). "Une caracterisation des fonctions sequentielles et des fonctions sous-sequentielles en tant que relations rationnelles". In: *Theoretical Computer Science* 5.3. ISSN: 0304-3975. URL: https://doi.org/10.1016/0304-3975(77)90049-4.

Chopin, Nicolas and Omiros Papaspiliopoulos (2020). *An Introduction to Sequential Monte Carlo*. Springer Series in Statistics. URL: https://doi.org/10.1007/978-3-030-47845-2.

Christensen, Peter Ebert, Vésteinn Snæbjarnarson, Andrea Dittadi, Serge Belongie, and Sagie Benaim (2024). "Assessing Neural Network Robustness via Adversarial Pivotal Tuning". In: *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*. WACV 2024 (oral). URL: https://openaccess.thecvf.com/content/WACV2024/html/Christensen_Assessing_Neural_Network_Robustness_via_Adversarial_Pivotal_Tuning_WACV_2024_paper.html.

Church, Kenneth Ward (2020). "Emerging trends: Subwords, seriously?" In: *Natural Language Engineering* 26.3. URL: https://doi.org/10.1017/S1351324920000145.

Clark, Thomas Hikaru, Moshe Poliak, Tamar Regev, A. J. Haskins, Caroline Robertson, and Edward Gibson (2025). "The Relationship Between Surprisal and Prosodic Prominence in Conversation Reflects Intelligibility-Oriented Pressures". In: *Cognitive Science* 49.10. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.70134.

Cognetta, Marco, David Pohl, Junyoung Lee, and Naoaki Okazaki (2025). "Pitfalls, Subtleties, and Techniques in Automata-Based Subword-Level Constrained Generation". In: *Tokenization Workshop*. URL: https://openreview.net/forum?id=DFybOGeGDS.

Cotterell, Ryan, Anej Svete, Clara Meister, Tianyu Liu, and Li Du (2024). *Formal Aspects of Language Modeling*. arXiv: 2311.04329 [cs.CL]. URL: https://arxiv.org/abs/2311.04329.

D'Antoni, Loris and Margus Veanes (2017). "The Power of Symbolic Automata and Transducers". In: *Computer Aided Verification*. URL: https://doi.org/10.1007/978-3-319-63387-9_3.

De Wulf, Martin, Laurent Doyen, Thomas A. Henzinger, and Jean-François Raskin (2006). "Antichains: A New Algorithm for Checking Universality of Finite Automata". In: *Proceedings of the International Conference on Computer Aided Verification*. URL: https://doi.org/10.1007/11817963_5.

Del Moral, Pierre, Arnaud Doucet, and Ajay Jasra (2006). "Sequential Monte Carlo samplers". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 68.3. URL: https://doi.org/10.1111/j.1467-9868.2006.00553.x.

Demberg, Vera and Frank Keller (2008). "Data from eye-tracking corpora as evidence for theories of syntactic processing complexity". In: *Cognition* 109.2. URL: https://www.sciencedirect.com/science/article/pii/S0010027708001741.

Word: A Cross-linguistic Typology (2002). URL: https://www.cambridge.org/core/books/word/7C775313B219D7B2566176502084 1D33.

Dolatian, Hossep and Jeffrey Heinz (2018). "Modeling Reduplication with 2-way Finite-State Transducers". In: *Proceedings of the Workshop on Computational Research in Phonetics, Phonology, and Morphology*. URL: https://aclanthology.org/W18-5807/.

Sequential Monte Carlo Methods in Practice (2001). Information Science and Statistics. ISBN: 978-1-4419-2887-0. URL: https://link.springer.com/book/10.1007/978-1-4757-3437-9.

Du, Kevin, Vésteinn Snæbjarnarson, Niklas Stoehr, Jennifer White, Aaron Schein, and Ryan Cotterell (2024). "Context versus Prior Knowledge in Language Models". In: *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2024. URL: https://aclanthology.org/2024.acl-long.714/.

Ehrlich, Susan F. and Keith Rayner (1981). "Contextual effects on word perception and eye movements during reading". In: *Journal of Verbal Learning and Verbal Behavior* 20.6. URL: https://www.sciencedirect.com/science/article/pii/S0022537181902206.

Farquhar, Sebastian, Jannik Kossen, Lorenz Kuhn, and Yarin Gal (2024). "Detecting hallucinations in large language models using semantic entropy". In: *Nature* 630.8017. ISSN: 1476-4687. URL: https://doi.org/10.1038/s41586-024-07421-0.

Fearnhead, Paul and Peter Clifford (2003). "On-Line Inference for Hidden Markov Models via Particle Filters". In: *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 65.4. URL: https://doi.org/10.1111/1467-9868.00421.

Frank, Stefan L., Leun J. Otten, Giulia Galli, and Gabriella Vigliocco (2013). "Word surprisal predicts N400 amplitude during reading". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. URL: https://aclanthology.org/P13-2152/.

— (2015). "The ERP response to the amount of information conveyed by words in sentences". In: *Brain and Language* 140. URL: https://www.sciencedirect.com/science/article/pii/S0093934X14001515.

Friðriksdóttir, Steinunn Rut, Annika Simonsen, Atli Snær Ásmundsson, Guðrún Lilja Friðjónsdóttir, Anton Karl Ingason, Vésteinn Snæbjarnarson, and Hafsteinn Einarsson (2024). "Ice and Fire: Dataset on Sentiment, Emotions, Toxicity, Sarcasm, Hate speech, Sympathy and More in Icelandic Blog Comments". In: *Proceedings of the Fourth Workshop on Threat, Aggression & Cyberbullying @ LREC-COLING-2024*. TRAC @ LREC-COLING 2024. URL: https://aclanthology.org/2024.trac-1.9/.

Futrell, Richard, Edward Gibson, and Roger P. Levy (2020). "Lossy-Context Surprisal: An Information-Theoretic Model of Memory Effects in Sentence Processing". In: *Cognitive Science* 44.3. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.12814.

Gage, Philip (1994). "A new algorithm for data compression". In: *The C Users Journal* 12.2. URL: https://dl.acm.org/doi/abs/10.5555/177910.177914.

Geh, Renato, Honghua Zhang, Kareem Ahmed, Benjie Wang, and Guy Van Den Broeck (2024). "Where is the signal in tokenization space?" In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2024.emnlp-main.230/.

Ghazvininejad, Marjan, Xing Shi, Yejin Choi, and Kevin Knight (2016). "Generating Topical Poetry". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/D16-1126/.

Giulianelli, Mario, Luca Malagutti, Juan Luis Gastaldi, Brian DuSell, Tim Vieira, and Ryan Cotterell (2024). "On the Proper Treatment of Tokenization in Psycholinguistics". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2024.emnlp-main.1032.

Giulianelli, Mario, Sarenne Wallbridge, and Raquel Fernández (2023). "Information Value: Measuring Utterance Predictability as Distance from Plausible Alternatives". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2023.emnlp-main.343/.

Goodkind, Adam and Klinton Bicknell (2018). "Predictive power of word surprisal for reading times is a linear function of language model quality". In: *Proceedings of the Workshop on Cognitive Modeling and Computational Linguistics*. URL: https://aclanthology.org/W18-0102.

Gorman, Kyle (2016). "Pynini: A Python library for weighted finite-state grammar compilation". In: *Proceedings of the SIGFSM Workshop on Statistical NLP and Weighted Automata*. URL: https://aclanthology.org/W16-2409/.

Grattafiori, Aaron et al. (2024). "The Llama 3 Herd of Models". In: *arXiv preprint arXiv:2407.21783*. URL: https://arxiv.org/abs/2407.21783.

Gruteke Klein, Keren, Yoav Meiri, Omer Shubi, and Yevgeni Berzak (2024). "The Effect of Surprisal on Reading Times in Information Seeking and Repeated Reading". In: *Proceedings of the Conference on Computational Natural Language Learning*. URL: https://aclanthology.org/2024.conll-1.17/.

Hale, John (2001). "A Probabilistic Earley Parser as a Psycholinguistic Model". In: *Second Meeting of the North American Chapter of the Association for Computational Linguistics*. URL: https://aclanthology.org/N01-1021.

Hamilton, Sil and David Mimno (2025). "Lost in Space: Finding the Right Tokens for Structured Output". In: *arXiv preprint arXiv:2502.14969*. URL: https://arxiv.org/abs/2502.14969.

Haspelmath, Martin (2011). "The Indeterminacy of Word Segmentation and the Nature of Morphology and Syntax". In: *Folia Linguistica* 45.1. URL: https://doi.org/10.1515/flin.2011.002.

Hayase, Jonathan, Alisa Liu, Noah A. Smith, and Sewoong Oh (2026). "Sampling from Your Language Model One Byte at a Time". In: *Forty-third International Conference on Machine Learning*. URL: https://openreview.net/forum?id=COQV7D1dAW.

Heinz, Jeffrey (2018). "The computational nature of phonological generalizations". In: *Phonological Typology*. URL: https://doi.org/10.1515/9783110451931-005.

Heng, Jeremy, Adrian N. Bishop, George Deligiannidis, and Arnaud Doucet (2020). "Controlled sequential Monte Carlo". In: *The Annals of Statistics* 48.5. URL: https://doi.org/10.1214/19-AOS1914.

Hill, Robin L. and Wayne S. Murray (2000). "Commas and Spaces: Effects of Punctuation on Eye Movements and Sentence Parsing". In: *Reading*

as a Perceptual Process. URL: https://www.sciencedirect.com/science/article/pii/B9780080436425500279.

Hinton, Geoffrey, Oriol Vinyals, and Jeff Dean (2015). "Distilling the Knowledge in a Neural Network". In: *arXiv preprint arXiv:1503.02531*. URL: https://arxiv.org/abs/1503.02531.

Hirotani, Masako, Lyn Frazier, and Keith Rayner (2006). "Punctuation and intonation effects on clause and sentence wrap-up: Evidence from eye movements". In: *Journal of Memory and Language* 54.3. URL: https://www.sciencedirect.com/science/article/pii/S0749596X05001440.

Hockett, Charles F. (1958). *A Course in Modern Linguistics*. URL: https://archive.org/details/courseinmodernli0000hock.

Hofmann, Valentin, Janet Pierrehumbert, and Hinrich Schütze (2021). "Superbizarre Is Not Superb: Derivational Morphology Improves BERT's Interpretation of Complex Words". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. URL: https://aclanthology.org/2021.acl-long.279/.

Holtzman, Ari, Peter West, Vered Shwartz, Yejin Choi, and Luke Zettlemoyer (2021). "Surface Form Competition: Why the Highest Probability Answer Isn't Always Right". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2021.emnlp-main.564/.

Hoover, Jacob Louis, Morgan Sonderegger, Steven T. Piantadosi, and Timothy J. O'Donnell (2023). "The Plausibility of Sampling as an Algorithmic Theory of Sentence Processing". In: *Open Mind* 7. URL: https://doi.org/10.1162/opmi_a_00086.

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman (2001). *Introduction to Automata Theory, Languages, and Computation*. 2nd. ISBN: 0201441241. URL: https://openlibrary.org/books/OL6795097M.

Hopton, Zachary William, Francesco Ignazio Re, Samuel Kiegeland, Andreas Opedal, Eleanor Chodroff, and Ryan Cotterell (2026). "Revisiting the Estimation of Unigram Surprisal". Under review.

Horvitz, D. G. and D. J. Thompson (1952). "A Generalization of Sampling Without Replacement from a Finite Universe". In: *Journal of the American Statistical Association* 47.260. URL: https://www.jstor.org/stable/2280784.

Ilyinskaya, Evgenia, Vésteinn Snæbjarnarson, Hanne Krage Carlsen, and Björn Oddsson (2024). "Brief communication: Small-scale geohazards cause significant and highly variable impacts on emotions". In: *Natural Hazards and Earth System Sciences* 24.9. NHESS 2024. URL: https://nhess.copernicus.org/articles/24/3115/2024/.

Ingólfsdóttir, Svanhvít Lilja, Pétur Ragnarsson, Haukur Jónsson, Haukur Símonarson, Vilhjálmur Þorsteinsson, and Vésteinn Snæbjarnarson (2023). "Byte-Level Grammatical Error Correction Using Synthetic and Curated Corpora". In: *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2023 (oral). URL: https://aclanthology.org/2023.acl-long.402/.

Jurafsky, Daniel and James H. Martin (2026). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd. Online manuscript released January 6, 2026. URL: https://web.stanford.edu/~jurafsky/slp3/.

Kahn, Arthur B. (1962). "Topological sorting of large networks". In: *Communications of the ACM* 5.11. URL: https://dl.acm.org/doi/10.1145/368996.369025.

Kahn, Herman (1955). *Use of Different Monte Carlo Sampling Techniques*. Paper P-766. RAND Corporation. URL: https://www.rand.org/pubs/papers/P766.html.

Kaplan, Ronald M. and Martin Kay (1994). "Regular Models of Phonological Rule Systems". In: *Computational Linguistics* 20.3. URL: https://aclanthology.org/J94-3001/.

Kennedy, Alan, Robin Hill, and Joël Pynte (2003). "The Dundee corpus". In: *Proceedings of the European Conference on Eye Movement*.

Kiegeland, Samuel, Vésteinn Snæbjarnarson, Tim Vieira, and Ryan Cotterell (2026). "On the Proper Treatment of Units in Surprisal Theory". In: *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. ACL 2026. URL: https://aclanthology.org/2026.acl-long.1485/.

Kliegl, Reinhold, Ellen Grabner, Martin Rolfs, and Ralf Engbert (2004). "Length, frequency, and predictability effects of words on eye movements in reading". In: *European Journal of Cognitive Psychology* 16.1-2. URL: https://doi.org/10.1080/09541440340000213.

König, Dénes (1927). "Über eine Schlussweise aus dem Endlichen ins Unendliche: Punktmengen. — Kartenfärben. — Verwandtschaftsbeziehungen. — Schachspiel". In: *Acta litterarum ac scientiarum Regiae Universitatis Hungaricae Francisco-Josephinae: Sectio scientiarum mathematicarum* 3. URL: https://acta.bibl.u-szeged.hu/13338/.

Koo, Terry, Frederick Liu, and Luheng He (2024). "Automata-based constraints for language model decoding". In: *Conference on Language Modeling*. URL: https://openreview.net/forum?id=BDBdblmyzY.

Koskenniemi, Kimmo (1983). "Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production". PhD thesis.

University of Helsinki. URL: https://researchportal.helsinki.fi/en/publications/two-level-morphology-a-general-computational-model-for-word-form-.

Kuhn, Lorenz, Yarin Gal, and Sebastian Farquhar (2023). "Semantic Uncertainty: Linguistic Invariances for Uncertainty Estimation in Natural Language Generation". In: *The Eleventh International Conference on Learning Representations*. URL: https://openreview.net/forum?id=VD-AYtP0dve.

Kuribayashi, Tatsuki, Yohei Oseki, and Timothy Baldwin (2024). "Psychometric Predictive Power of Large Language Models". In: *Findings of the Association for Computational Linguistics: NAACL*. URL: https://aclanthology.org/2024.findings-naacl.129/.

Kuribayashi, Tatsuki, Yohei Oseki, Souhaib Ben Taieb, Kentaro Inui, and Timothy Baldwin (2025). "Large Language Models Are Human-Like Internally". In: *Transactions of the Association for Computational Linguistics* 13. URL: https://doi.org/10.1162/TACL.a.58.

Kwon, Woosuk, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica (2023). "Efficient Memory Management for Large Language Model Serving with PagedAttention". In: *Proceedings of the Symposium on Operating Systems Principles*. URL: https://doi.org/10.1145/3600006.3613165.

Lang, Nico, Vésteinn Snæbjarnarson, Elijah Cole, Oisin Mac Aodha, Christian Igel, and Serge Belongie (2024). "From Coarse to Fine-Grained Open-Set Recognition". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR 2024. URL: https://openaccess.thecvf.com/content/CVPR2024/html/Lang_From_Coarse_to_Fine-Grained_Open-Set_Recognition_CVPR_2024_paper.html.

Lau, Jey Han, Alexander Clark, and Shalom Lappin (2017). "Grammaticality, Acceptability, and Probability: A Probabilistic View of Linguistic Knowledge". In: *Cognitive Science* 41.5. URL: https://onlinelibrary.wiley.com/doi/abs/10.1111/cogs.12414.

Leviathan, Yaniv, Matan Kalman, and Yossi Matias (2023). "Fast Inference from Transformers via Speculative Decoding". In: *Proceedings of the 40th International Conference on Machine Learning*. URL: https://proceedings.mlr.press/v202/leviathan23a.html.

Levy, Roger (2008). "Expectation-based syntactic comprehension". In: *Cognition* 106.3. URL: https://doi.org/10.1016/j.cognition.2007.05.006.

Lew, Alexander K., Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka (2023). "Sequential Monte Carlo Steering of Large Language Models using Probabilistic Programs". In: *ICML Workshop: Sampling and Optimization in Discrete Space*. URL: https://openreview.net/forum?id=Ul2K0qXxXy.

Li, Xiang Lisa, Ari Holtzman, Daniel Fried, Percy Liang, Jason Eisner, Tatsunori Hashimoto, Luke Zettlemoyer, and Mike Lewis (2023). "Contrastive Decoding: Open-ended Text Generation as Optimization". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. URL: https://aclanthology.org/2023.acl-long.687/.

Liberman, Alvin M., Franklin S. Cooper, Donald P. Shankweiler, and Michael Studdert-Kennedy (1967). "Perception of the Speech Code". In: *Psychological Review* 74.6. URL: https://doi.org/10.1037/h0020279.

Lin, Chu-Cheng, Hao Zhu, Matthew R. Gormley, and Jason Eisner (2019). "Neural Finite-State Transducers: Beyond Rational Relations". In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. URL: https://aclanthology.org/N19-1024/.

Lipkin, Benjamin, Benjamin LeBrun, Jacob Hoover Vigly, João Loula, David R. MacIver, Li Du, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Timothy J. O'Donnell, Alexander K. Lew, and Tim Vieira (2025). "Fast Controlled Generation from Language Models with Adaptive Weighted Rejection Sampling". In: *Conference on Language Modeling*. URL: https://arxiv.org/abs/2504.05410.

Liu, Alisa, Maarten Sap, Ximing Lu, Swabha Swayamdipta, Chandra Bhagavatula, Noah A. Smith, and Yejin Choi (2021). "DExperts: Decoding-Time Controlled Text Generation with Experts and Anti-Experts". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. URL: https://aclanthology.org/2021.acl-long.522/.

Loeschcke, Sebastian, Mads Toftrup, Michael J. Kastoryano, Serge Belongie, and Vésteinn Snæbjarnarson (2024). "LoQT: Low-Rank Adapters for Quantized Pretraining". In: *Advances in Neural Information Processing Systems*. Vol. 37. NeurIPS 2024. URL: https://proceedings.neurips.cc/paper_files/paper/2024/hash/d0c6bc641a56bebee9d985b937307367-Abstract-Conference.html.

Loula, João, Benjamin LeBrun, Li Du, Ben Lipkin, Clemente Pasti, Gabriel Grand, Tianyu Liu, Yahya Emara, Marjorie Freedman, Jason Eisner, Ryan Cotterell, Vikash Mansinghka, Alexander K. Lew, Tim Vieira, and Timothy J. O'Donnell (2025). "Syntactic and Semantic Control of Large Language Models via Sequential Monte Carlo". In: *The Thirteenth International Conference on Learning Representations*. URL: https://arxiv.org/abs/2504.13139.

Luke, Steven G. and Kiel Christianson (2018). "The Provo corpus: A large eye-tracking corpus with predictability norms". In: *Behavior Research Methods* 50.2. URL: https://link.springer.com/article/10.3758/s13428-017-0908-4.

Lundberg, Scott and Marco Tulio Ribeiro (2023). *The Art of Prompt Design: Prompt Boundaries and Token Healing*. Medium post. URL: https://medium.com/data-science/the-art-of-prompt-design-prompt-boundaries-and-token-healing-3b2448b0be38.

Madow, William G. (1949). "On the Theory of Systematic Sampling, II". In: *The Annals of Mathematical Statistics* 20.3. URL: https://doi.org/10.1214/aoms/1177729988.

Marcus, Mitchell P., Beatrice Santorini, and Mary Ann Marcinkiewicz (1993). "Building a Large Annotated Corpus of English: The Penn Treebank". In: *Computational Linguistics*. URL: https://aclanthology.org/J93-2004/.

Marr, David (1982). *Vision: A Computational Investigation into the Human Representation and Processing of Visual Information*. ISBN: 9780716712848. URL: https://search.worldcat.org/title/301016436.

Meister, Clara, Afra Amini, Tim Vieira, and Ryan Cotterell (2021a). "Conditional Poisson Stochastic Beam Search". In: *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. URL: https://aclanthology.org/2021.emnlp-main.52.

Meister, Clara, Tiago Pimentel, Patrick Haller, Lena Jäger, Ryan Cotterell, and Roger Levy (2021b). "Revisiting the Uniform Information Density Hypothesis". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2021.emnlp-main.74/.

Merity, Stephen, Caiming Xiong, James Bradbury, and Richard Socher (2017). "Pointer Sentinel Mixture Models". In: *International Conference on Learning Representations*. URL: https://openreview.net/forum?id=Byj72udxe.

Miller, George A. and Kathryn Ojemann McKean (1964). "A Chronometric Study of Some Relations between Sentences". In: *Quarterly Journal of Experimental Psychology* 16.4. URL: https://doi.org/10.1080/17470216408416385.

Mitchell, Jeff, Mirella Lapata, Vera Demberg, and Frank Keller (2010). "Syntactic and Semantic Factors in Processing Difficulty: An Integrated Measure". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics*. URL: https://aclanthology.org/P10-1021/.

Mohri, Mehryar (1997). "Finite-State Transducers in Language and Speech Processing". In: *Computational Linguistics*. URL: https://aclanthology.org/J97-2003/.

— (2009). "Weighted Automata Algorithms". In: *Handbook of Weighted Automata*. Chap. 6. ISBN: 978-3-642-01492-5. URL: https://doi.org/10.1007/978-3-642-01492-5_6.

Murphy, Elliot (2024). "What is a word?" In: *arXiv preprint arXiv:2402.12605*. URL: https://arxiv.org/abs/2402.12605.

Naesseth, Christian A., Fredrik Lindsten, and Thomas B. Schön (2019). "Elements of Sequential Monte Carlo". In: *Foundations and Trends in Machine Learning* 12.3. URL: https://doi.org/10.1561/2200000074.

Nair, Sathvik and Philip Resnik (2023). "Words, Subwords, and Morphemes: What Really Matters in the Surprisal-Reading Time Relationship?" In: *Findings of the Association for Computational Linguistics: EMNLP*. URL: https://aclanthology.org/2023.findings-emnlp.752/.

Nivre, Joakim, Daniel Zeman, Filip Ginter, and Francis Tyers (2017). "Universal Dependencies". In: *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics: Tutorial Abstracts*. URL: https://aclanthology.org/E17-5001/.

Noord, Gertjan van and Dale Gerdemann (2001). "Finite State Transducers with Predicates and Identities". In: *Grammars* 4.3. URL: https://doi.org/10.1023/A:1012291501330.

Oh, Byung-Doh, Christian Clark, and William Schuler (2021). "Surprisal Estimators for Human Reading Times Need Character Models". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics and the International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. URL: https://aclanthology.org/2021.acl-long.290/.

Oh, Byung-Doh and William Schuler (2023). "Why Does Surprisal From Larger Transformer-Based Language Models Provide a Poorer Fit to Human Reading Times?" In: *Transactions of the Association for Computational Linguistics* 11. URL: https://aclanthology.org/2023.tacl-1.20/.

— (2024). "Leading Whitespaces of Language Models' Subword Vocabulary Pose a Confound for Calculating Word Probabilities". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2024.emnlp-main.202/.

— (2025). "The Impact of Token Granularity on the Predictive Power of Language Model Surprisal". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. URL: https://aclanthology.org/2025.acl-long.209/.

Oncina, J., P. Garcia, and E. Vidal (1993). "Learning subsequential transducers for pattern recognition interpretation tasks". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 15.5. URL: https://ieeexplore.ieee.org/document/211465.

Opedal, Andreas, Eleanor Chodroff, Ryan Cotterell, and Ethan Wilcox (2024). "On the Role of Context in Reading Time Prediction". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2024.emnlp-main.179/.

OpenAI (2024). *Tiktoken: A Fast BPE Tokeniser for OpenAI's Models*. URL: https://github.com/openai/tiktoken.

Orwell, George (1949). *Nineteen Eighty-Four*. URL: https://www.orwellfoundation.com/the-orwell-foundation/orwell/books-by-orwell/nineteen-eighty-four/.

Phan, Buu, Brandon Amos, Itai Gat, Marton Havasi, Matthew J. Muckley, and Karen Ullrich (2025). "Exact Byte-Level Probabilities from Tokenized Language Models for FIM-Tasks and Model Ensembles". In: *The Thirteenth International Conference on Learning Representations*. URL: https://proceedings.iclr.cc/paper_files/paper/2025/hash/5ed5c3c846f684a54975ad7a2525199f-Abstract-Conference.html.

Phan, Buu, Marton Havasi, Matthew Muckley, and Karen Ullrich (2024). "Understanding and Mitigating Tokenization Bias in Language Models". In: *arXiv preprint arXiv:2406.16829*. URL: https://arxiv.org/abs/2406.16829.

Pimentel, Tiago and Clara Meister (2024). "How to Compute the Probability of a Word". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2024.emnlp-main.1020/.

Handbook of Automata Theory (2021). ISBN: 978-3-98547-006-8. URL: https://doi.org/10.4171/Automata.

Pin, Jean-Éric (2025). *Mathematical Foundations of Automata Theory*. URL: https://www.irif.fr/~jep/PDF/MPRI/MPRI.pdf.

Pinker, Steven (1994). *The Language Instinct*. URL: https://archive.org/details/languageinstinct00pink.

Rabin, Michael O. and Dana Scott (1959). "Finite automata and their decision problems". In: *IBM Journal of Research and Development* 3.2. URL: https://ieeexplore.ieee.org/document/5392601.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever (2019). "Language Models are Unsupervised Multitask Learners". In: *OpenAI Blog*. URL: https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Rastogi, Pushpendre, Ryan Cotterell, and Jason Eisner (2016). "Weighting Finite-State Transductions With Neural Context". In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. URL: https://aclanthology.org/N16-1076/.

Ravfogel, Shauli, Anej Svete, Vésteinn Snæbjarnarson, and Ryan Cotterell (2025). "Gumbel Counterfactual Generation From Language Models". In: *The Thirteenth International Conference on Learning Representations*. ICLR 2025. URL: https://openreview.net/forum?id=TUC0ZT2zIQ.

Rayner, Keith (1975). "Parafoveal identification during a fixation in reading". In: *Acta Psychologica* 39.4. URL: https://www.sciencedirect.com/science/article/pii/0001691875900116.

— (1998). "Eye movements in reading and information processing: 20 years of research." In: *Psychological Bulletin* 124.3. URL: https://doi.org/10.1037/0033-2909.124.3.372.

Rayner, Keith, Marcia Carlson, and Lyn Frazier (1983). "The interaction of syntax and semantics during sentence processing: eye movements in the analysis of semantically biased sentences". In: *Journal of Verbal Learning and Verbal Behavior* 22.3. URL: https://www.sciencedirect.com/science/article/pii/S0022537183902360.

Rayner, Keith, Gretchen Kambe, and Susan A. Duffy (2000). "The effect of clause wrap-up on eye movements during reading". In: *The Quarterly Journal of Experimental Psychology Section A* 53.4. URL: https://doi.org/10.1080/713755934.

Rayner, Keith and Gary E. Raney (1996). "Eye movement control in reading and visual search: Effects of word frequency". In: *Psychonomic Bulletin & Review* 3.2. URL: https://doi.org/10.3758/BF03212426.

Rayner, Keith, Arnold D. Well, Alexander Pollatsek, and James H. Bertera (1982). "The availability of useful information to the right of fixation in reading". In: *Perception & Psychophysics* 31.6. URL: https://doi.org/10.3758/BF03204186.

Re, Francesco Ignazio, Andreas Opedal, Glib Manaiev, Mario Giulianelli, and Ryan Cotterell (2025). "A Spatio-Temporal Point Process for Fine-Grained Modeling of Reading Behavior". In: *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. URL: https://aclanthology.org/2025.acl-long.1474/.

Riley, Michael, Cyril Allauzen, and Martin Jansche (2009). "OpenFst: An Open-Source, Weighted Finite-State Transducer Library and its Applications to Speech and Language". In: *Proceedings of Human Language Technologies: The 2009 Annual Conference of the North American Chapter of the Association for Computational Linguistics, Companion Volume: Tutorial Abstracts*. URL: https://aclanthology.org/N09-4005/.

Runcie, Nicholas T., Charlotte M. Deane, and Fergus Imrie (2026). "Assessing the Chemical Intelligence of Large Language Models". In: *Journal of Chemical Information and Modeling* 66.1. URL: https://doi.org/10.1021/acs.jcim.5c02145.

Saussure, Ferdinand de (1997). *Saussure's Second Course of Lectures on General Linguistics (1908–1909): From the Notebooks of Albert Riedlinger and Charles Patois*. URL: https://shop.elsevier.com/books/saussures-

second-course-of-lectures-on-general-linguistics-1908-09/wolf/978-0-08-042579-5.

Schotter, Elizabeth R., Bernhard Angele, and Keith Rayner (2012). "Parafoveal processing in reading". In: *Attention, Perception, & Psychophysics* 74.1. URL: https://doi.org/10.3758/s13414-011-0219-2.

Schotter, Elizabeth R. and Brian Dillon (2025). "A beginner's guide to eye tracking for psycholinguistic studies of reading". In: *Behavior Research Methods* 57.2. URL: https://doi.org/10.3758/s13428-024-02572-4.

Schwartz, Idan, Vésteinn Snæbjarnarson, Hila Chefer, Serge Belongie, Lior Wolf, and Sagie Benaim (2023). "Discriminative Class Tokens for Text-to-Image Diffusion Models". In: *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. ICCV 2023. URL: https://openaccess.thecvf.com/content/ICCV2023/html/Schwartz_Discriminative_Class_Tokens_for_Text-to-Image_Diffusion_Models_ICCV_2023_paper.html.

Sennrich, Rico, Barry Haddow, and Alexandra Birch (2016). "Neural Machine Translation of Rare Words with Subword Units". In: *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. URL: https://aclanthology.org/P16-1162/.

Shah, Rohan and Dirk P. Kroese (2018). "Without-replacement sampling for particle methods on finite state spaces". In: *Statistics and Computing* 28.3. ISSN: 1573-1375. URL: https://doi.org/10.1007/s11222-017-9752-8.

Shain, Cory (2019). "A large-scale study of the effects of word frequency and predictability in naturalistic reading". In: *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*. URL: https://aclanthology.org/N19-1413/.

— (2024). "Word Frequency and Predictability Dissociate in Naturalistic Reading". In: *Open Mind* 8. URL: https://doi.org/10.1162/opmi_a_00119.

Shain, Cory, Clara Meister, Tiago Pimentel, Ryan Cotterell, and Roger Levy (2024). "Large-scale evidence for logarithmic effects of word predictability on reading time". In: *Proceedings of the National Academy of Sciences* 121.10. URL: https://www.pnas.org/doi/abs/10.1073/pnas.2307876121.

Siegelman, Noam, Sascha Schroeder, Cengiz Acartürk, Hee-Don Ahn, Svetlana Alexeeva, Simona Amenta, Raymond Bertram, Rolando Bonandrini, Marc Brysbaert, Daria Chernova, et al. (2022). "Expanding horizons of cross-linguistic research on reading: The Multilingual Eye-movement Corpus (MECO)". In: *Behavior Research Methods* 54.6. URL: https://link.springer.com/article/10.3758/s13428-021-01772-6.

Simonsen, Annika, Barbara Scalvini, Uni Johannesen, Iben Nyholm Debess, Hafsteinn Einarsson, and Vésteinn Snæbjarnarson (2026). "BRAGD: Con-

strained Multi-Label POS Tagging for Faroese". In: *Proceedings of the 15th Language Resources and Evaluation Conference (LREC 2026)*. LREC 2026. URL: https://aclanthology.org/2026.lrec-1.913/.

Smith, Nathaniel J. and Roger Levy (2013). "The effect of word predictability on reading time is logarithmic". In: *Cognition* 128.3. URL: https://www.sciencedirect.com/science/article/pii/S0010027713000413.

Snæbjarnarson, Vésteinn, Kevin Du, Niklas Stoehr, Serge Belongie, Ryan Cotterell, Nico Lang, and Stella Frank (2025). "Taxonomy-Aware Evaluation of Vision-Language Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. CVPR 2025. URL: https://openaccess.thecvf.com/content/CVPR2025/html/Snaebjarnarson_Taxonomy-Aware_Evaluation_of_Vision-Language_Models_CVPR_2025_paper.html.

Snæbjarnarson, Vésteinn, Samuel Kiegeland, Tianyu Liu, Reda Boumasmoud, Ryan Cotterell, and Tim Vieira (2026a). "Transducing Language Models". In: *The Fourteenth International Conference on Learning Representations*. ICLR 2026. URL: https://openreview.net/forum?id=qOyF214xmg.

Snæbjarnarson, Vésteinn, Samuel Kiegeland, Manuel de Prada Corral, Ryan Cotterell, and Tim Vieira (2026b). "Stochastic Estimation of Transduced Language Models". In: *arXiv preprint arXiv:2608.27428*. URL: https://arxiv.org/abs/2608.27428.

Snæbjarnarson, Vésteinn, Annika Simonsen, Goran Glavaš, and Ivan Vulić (2023). "Transfer to a Low-Resource Language via Close Relatives: The Case Study on Faroese". In: *Proceedings of the 24th Nordic Conference on Computational Linguistics (NoDaLiDa)*. NoDaLiDa 2023. URL: https://aclanthology.org/2023.nodalida-1.74/.

Snæbjarnarson, Vésteinn, Anej Svete, Josef Valvoda, Reda Boumasmoud, Brian DuSell, and Ryan Cotterell (2026c). "Causally Evaluating the Learnability of Formal Language Tasks". In: *Forty-third International Conference on Machine Learning*. ICML 2026. URL: https://openreview.net/forum?id=sYUCXUeApn.

Sohoglu, Ediz, Loes Beckers, and Matthew H. Davis (2024). "Convergent neural signatures of speech prediction error are a biological marker for spoken word recognition". In: *Nature Communications* 15.1. URL: https://doi.org/10.1038/s41467-024-53782-5.

Song, Xinying, Alex Salcianu, Yang Song, Dave Dopson, and Denny Zhou (2021). "Fast WordPiece Tokenization". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2021.emnlp-main.160/.

Speer, Robyn (2022). *rspeer/wordfreq: v3.0*. URL: https://doi.org/10.5281/zenodo.7199437.

Stahlberg, Felix, Christopher Bryant, and Bill Byrne (2019). "Neural Grammatical Error Correction with Finite State Transducers". In: *Proceedings of the Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. URL: https://aclanthology.org/N19-1406/.

Stoehr, Niklas, Kevin Du, Vésteinn Snæbjarnarson, Robert West, Ryan Cotterell, and Aaron Schein (2024). "Activation Scaling for Steering and Interpreting Language Models". In: *Findings of the Association for Computational Linguistics: EMNLP 2024*. Findings of EMNLP 2024. URL: https://aclanthology.org/2024.findings-emnlp.479/.

Tezcan, Filiz, Hugo Weissbart, and Andrea E Martin (2023). "A tradeoff between acoustic and linguistic feature encoding in spoken language comprehension". In: *eLife* 12. URL: https://doi.org/10.7554/eLife.82386.

Tsipidi, Eleftheria, Samuel Kiegeland, Franz Nowak, Tianyang Xu, Ethan Wilcox, Alex Warstadt, Ryan Cotterell, and Mario Giulianelli (2025). "The Harmonic Structure of Information Contours". In: *Proceedings of the Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. URL: https://aclanthology.org/2025.acl-long.1527/.

Tsipidi, Eleftheria, Franz Nowak, Ryan Cotterell, Ethan Wilcox, Mario Giulianelli, and Alex Warstadt (2024). "Surprise! Uniform Information Density Isn't the Whole Story: Predicting Surprisal Contours in Long-form Discourse". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. URL: https://aclanthology.org/2024.emnlp-main.1047/.

Ulm, Jannek, Kevin Du, and Vésteinn Snæbjarnarson (2025). "Contrastive Decoding for Synthetic Data Generation in Low-Resource Language Modeling". In: *Proceedings of the First BabyLM Workshop*. BabyLM @ EMNLP 2025. URL: https://aclanthology.org/2025.babylm-main.2/.

Veanes, Margus, Pieter Hooimeijer, Benjamin Livshits, David Molnar, and Nikolaj Bjørner (2012). "Symbolic finite state transducers: Algorithms and applications". In: *Proceedings of the Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*. URL: https://doi.org/10.1145/2103656.2103674.

Vieira, Tim, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Brian DuSell, John Terilla, Timothy J. O'Donnell, and Ryan Cotterell (2025a). "From Language Models over Tokens to Language Models over Characters". In: *Proceedings of the 42nd International Conference on Machine Learning*. URL: https://proceedings.mlr.press/v267/vieira25a.html.

Vieira, Tim, Tianyu Liu, Clemente Pasti, Yahya Emara, Brian DuSell, Benjamin LeBrun, Mario Giulianelli, Juan Luis Gastaldi, Timothy J. O'Don-

nell, and Ryan Cotterell (2025b). "Language Models over Canonical Byte-Pair Encodings". In: *Forty-second International Conference on Machine Learning*. URL: https://openreview.net/forum?id=eCVrfVDNSY.

Wallbridge, Sarenne, Peter Bell, and Catherine Lai (2023). "Do dialogue representations align with perception? An empirical study". In: *Proceedings of the Conference of the European Chapter of the Association for Computational Linguistics*. URL: https://aclanthology.org/2023.eacl-main.198/.

Wallbridge, Sarenne Carrol, Catherine Lai, and Peter Bell (2022). "Investigating perception of spoken dialogue acceptability through surprisal". In: *Annual Conference of the International Speech Communication Association, Interspeech*. URL: https://doi.org/10.21437/Interspeech.2022-10808.

Wang, Xinpeng, Bolei Ma, Chengzhi Hu, Leon Weber-Genzel, Paul Röttger, Frauke Kreuter, Dirk Hovy, and Barbara Plank (2024). ""My Answer is C": First-Token Probabilities Do Not Match Text Answers in Instruction-Tuned Language Models". In: *Findings of the Association for Computational Linguistics: ACL 2024*. URL: https://aclanthology.org/2024.findings-acl.441/.

Whiteley, Nick and Anthony Lee (2014). "Twisted Particle Filters". In: *The Annals of Statistics* 42.1. URL: https://doi.org/10.1214/13-AOS1167.

Wilcox, Ethan G., Tiago Pimentel, Clara Meister, Ryan Cotterell, and Roger P. Levy (2023). "Testing the Predictions of Surprisal Theory in 11 Languages". In: *Transactions of the Association for Computational Linguistics* 11. URL: https://aclanthology.org/2023.tacl-1.82/.

Wilcox, Ethan Gotlieb, Jon Gauthier, Jennifer Hu, Peng Qian, and Roger Levy (2020). "On the predictive power of neural language models for human real-time comprehension behavior". In: *Proceedings of the Annual Meeting of the Cognitive Science Society*. URL: https://escholarship.org/uc/item/738338tm.

Willard, Brandon T. and Rémi Louf (2023). "Efficient Guided Generation for Large Language Models". In: *arXiv preprint arXiv:2307.09702*. URL: https://arxiv.org/abs/2307.09702.

Wolf, Thomas, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Remi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander Rush (2020). "Transformers: State-of-the-Art Natural Language Processing". In: *Proceedings of the Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. URL: https://aclanthology.org/2020.emnlp-demos.6.

Wood, Simon N. (2017). *Generalized Additive Models: An Introduction with R*. 2nd ed. URL: https://doi.org/10.1201/9781315370279.

Xefteri, Vicky, Tim Vieira, Ryan Cotterell, and Afra Amini (2025). "Syntactic Control of Language Models by Posterior Inference". In: *Findings of the Association for Computational Linguistics: ACL 2025*. URL: https://aclanthology.org/2025.findings-acl.1300/.

Xu, Weijie, Jason Chon, Tianran Liu, and Richard Futrell (2023). "The Linearity of the Effect of Surprisal on Reading Times across Languages". In: *Findings of the Association for Computational Linguistics: EMNLP*. URL: https://aclanthology.org/2023.findings-emnlp.1052/.

Xue, Linting, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel (2022). "ByT5: Towards a Token-Free Future with Pre-trained Byte-to-Byte Models". In: *Transactions of the Association for Computational Linguistics* 10. URL: https://aclanthology.org/2022.tacl-1.17/.

Zhao, Stephen, Rob Brekelmans, Alireza Makhzani, and Roger Grosse (2024). "Probabilistic Inference in Language Models via Twisted Sequential Monte Carlo". In: *Proceedings of the 41st International Conference on Machine Learning*. URL: https://proceedings.mlr.press/v235/zhao24c.html.