STOCHASTIC ESTIMATION OF TRANSDUCED LANGUAGE MODELS

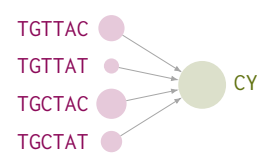
Anonymous authors
Paper under double-blind review

ABSTRACT

Transduced language models (TLMs) compose a pretrained *source* language model with a functional finite-state transducer to induce a model over the *target* strings of the transducer. Computing the probability of a target prefix under a TLM amounts to summing the source-model probabilities of all strings that map to strings beginning with that target prefix. This set can be infinitely large, so prior work uses a computational shortcut and biased pruning to make the breadth-first search feasible. We instead propose an algorithm that samples source prefixes without replacement, reweighting each prefix by the inverse of its inclusion probability. Applying this correction gives an unbiased estimator of the TLM's prefix probability and lets us estimate the mass lost by pruning. The algorithm reduces the number of prefixes as more probability mass is added to the running estimate. This can save computation and guarantees that the run halts with probability one. The algorithm achieves a better compute–variance tradeoff on English text when converting a model over subwords to a model over whole words, and lower error on DNA-to-amino-acid transduction than other sequential Monte Carlo baselines. On the DNA-to-amino-acid transduction, it reduces runtime by orders of magnitude and makes scoring long target strings feasible. Repeating a published reading-time analysis substantially lowers estimated corpus surprisal while confirming its conclusions.

1 INTRODUCTION

Suppose we want to reuse a language model trained on *DNA* to score *amino-acid sequences*, rather than train a new model. A standard conversion defines a many-to-one mapping from DNA triplets, called *codons*, to amino acids. We can then ask, what is the probability that the language model's output, after translation, begins with the amino acids `CY`? Each of these amino acids has two possible codons, so four six-base DNA prefixes encode `CY`. To score `CY`, we sum the probabilities of these four prefixes.



Consider the nine-amino-acid sequence `CYIQNCPLG` on the right, which has 9,216 possible DNA encodings. Proteins commonly contain hundreds of amino acids and can extend to thousands. Even with only two codon choices at each position, a 300-amino-acid sequence would have far more encodings than the number of atoms in the observable universe. When probability mass is spread out, recovering most of it by enumeration is infeasible.

```
                                CCG TTG GGG
        ATC                     CCA TTA GGA
TGC TAT ATT CAG AAC TGC CCC CTG GGC
TGT TAC ATA CAA AAT TGT CCT CTA GGT
 ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓   ↓
 C   Y   I   Q   N   C   P   L   G
```

$2 \times 2 \times 3 \times 2 \times 2 \times 2 \times 4 \times 6 \times 4$

This is an instance of the *string mismatch problem* studied by Snæbjarnarson et al. (2026). Their *transduced language models* (TLMs) compose a pretrained *source* model $p_{\mathcal{X}}$ with a functional finite-state transducer implementing a transformation f, inducing a distribution over *target* strings. The core quantity in a TLM is the *prefix probability*, the probability that its output begins with a specified string $\boldsymbol{y}$. Writing $\boldsymbol{y} \preceq f(\boldsymbol{x})$ for this condition and $\mathcal{P}(\boldsymbol{y})$ for the set of source strings satisfying it, we define

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

The set $\mathcal{P}(\boldsymbol{y})$ is called the *precover* and can contain infinitely many strings. The transduced model's next-symbol probabilities follow by taking ratios, $\overrightarrow{p_{\mathcal{Y}}}(y \mid \boldsymbol{y}) = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}y)/\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

Snæbjarnarson et al. (2026) derive a computational shortcut that uses source-prefix probabilities to sum over sets of source strings at once. To make this computation practical, they use threshold pruning to retain the most likely prefixes accounting for at least $1 - \tau$ of the current candidate mass. This can work well when probability mass is concentrated on relatively few prefixes (Snæbjarnarson et al., 2026; Vieira et al., 2025). The prior evaluation compares normalized next-symbol distributions across thresholds, which does not quantify how much prefix probability pruning discards.

This work replaces threshold pruning with *resampling without replacement*. We sample distinct source prefixes and divide each retained weight by its probability of inclusion (Horvitz & Thompson, 1952; Fearnhead & Clifford, 2003; Shah & Kroese, 2018). We incorporate this approach into the recursive TLM computation, showing that the corrections give an unbiased estimate of each target prefix probability whenever the run halts with probability one (Theorem 3.1). To ensure almost-sure halting and speed up computation, we propose an algorithm that uses the mass already accumulated to reduce the number of retained prefixes, together with a randomized stopping mechanism that preserves unbiasedness (Proposition 3.2). The difference between the mean of the unbiased estimates and the threshold-pruned value then gives an estimate of the prefix mass lost through pruning.

Under a bigram DNA model, threshold-pruned beam summing exceeds a five-minute limit for a sequence of length 13, whereas our adaptive method can handle 200 amino-acids in about five seconds (Fig. 6). With an additional cap on the number of retained prefixes, beam summing finishes but returns zero probability from position 17. Exact probabilities validate our estimates for the bigram models and short neural-DNA targets (§4). We compare our estimator against two unbiased sequential Monte Carlo (SMC) baselines that extend each retained prefix by one sampled live source symbol and use resampling with replacement. Our method achieves a better compute–variance tradeoff on encyclopedic text and lower error at the same maximum particle count on DNA. Finally, replacing threshold pruning in the TLM-based reading-time analysis of Kiegeland et al. (2026) lowers estimated corpus surprisal by 106 nats without changing the tested conclusions (§4.3). This lets us assess how pruning bias affects the analysis. The connection to related work is discussed in App. L.

2 Background

We review the background and notation for language models and transduced language models, largely following the definitions of Snæbjarnarson et al. (2026).[1]

Let $\mathcal{X}$ be a set of symbols called an **alphabet**, and $\mathcal{X}^*$ be the set of all finite strings of those symbols. We write $\boldsymbol{x}' \preceq \boldsymbol{x}$, equivalently $\boldsymbol{x} \succeq \boldsymbol{x}'$, when $\boldsymbol{x}'$ is a prefix of $\boldsymbol{x}$. A **cylinder** with basis $\boldsymbol{x}$ is $\langle \boldsymbol{x} \rangle \stackrel{\text{def}}{=} \{\boldsymbol{x}\boldsymbol{x}' \mid \boldsymbol{x}' \in \mathcal{X}^*\}$. We define the **prefix-free** operation $\mathrm{pf}(Z)$ as $\mathrm{pf}(Z) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in Z \colon \nexists \boldsymbol{x}' \in Z \text{ such that } \boldsymbol{x}' \prec \boldsymbol{x}\}$. A **language model** $p_\mathcal{X}$ is a distribution over $\mathcal{X}^*$. With $\textsc{eos} \notin \mathcal{X}$ a special **end-of-sequence symbol**, we define the **prefix probability** and the **conditional prefix probabilities** as

$$\overrightarrow{p_\mathcal{X}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{\boldsymbol{x}' \in \mathcal{X}^*} p_\mathcal{X}(\boldsymbol{x}\boldsymbol{x}'), \qquad \overrightarrow{p_\mathcal{X}}(\boldsymbol{x}' \mid \boldsymbol{x}) \stackrel{\text{def}}{=} \frac{\overrightarrow{p_\mathcal{X}}(\boldsymbol{x}\boldsymbol{x}')}{\overrightarrow{p_\mathcal{X}}(\boldsymbol{x})}, \qquad \overrightarrow{p_\mathcal{X}}(\textsc{eos} \mid \boldsymbol{x}) \stackrel{\text{def}}{=} \frac{p_\mathcal{X}(\boldsymbol{x})}{\overrightarrow{p_\mathcal{X}}(\boldsymbol{x})}. \tag{2}$$

When $\overrightarrow{p_\mathcal{X}}(\boldsymbol{x}) = 0$, we take $\overrightarrow{p_\mathcal{X}}(\boldsymbol{x}' \mid \boldsymbol{x}) = 0$ and $\overrightarrow{p_\mathcal{X}}(\textsc{eos} \mid \boldsymbol{x}) \stackrel{\text{def}}{=} 1$. An interface to the language model is then given by $p_\mathcal{X}(\boldsymbol{x}) = \overrightarrow{p_\mathcal{X}}(\textsc{eos} \mid \boldsymbol{x}) \prod_{t=1}^{|\boldsymbol{x}|} \overrightarrow{p_\mathcal{X}}(\boldsymbol{x}_t \mid \boldsymbol{x}_{<t})$.[2] When $\overrightarrow{p_\mathcal{X}}(\boldsymbol{x}) > 0$, the conditionals also give the distribution over the **completions** of a prefix, $p_\mathcal{X}(\boldsymbol{x}'' \mid \boldsymbol{x}) \stackrel{\text{def}}{=} p_\mathcal{X}(\boldsymbol{x}\boldsymbol{x}'')/\overrightarrow{p_\mathcal{X}}(\boldsymbol{x})$ for $\boldsymbol{x}'' \in \mathcal{X}^*$, written $\boldsymbol{x}'' \sim p_\mathcal{X}(\cdot \mid \boldsymbol{x})$. A **transduced language model** is a language model $p_\mathcal{Y}$ characterized by the tuple $(p_\mathcal{X}, \mathtt{f})$ where $\mathtt{f}$ is a transducer encoding a function $f \colon \mathcal{X}^* \to \mathcal{Y}^*$. Let X be a random variable drawn from $p_\mathcal{X}$. For $\boldsymbol{y} \in \mathcal{Y}^*$, we then define

$$p_\mathcal{Y}(\boldsymbol{y}) \stackrel{\text{def}}{=} \Pr_{X \sim p_\mathcal{X}}[\boldsymbol{y} = f(X)] = \sum_{\boldsymbol{x} \in \mathcal{X}^*} \mathbb{1}\{f(\boldsymbol{x}) = \boldsymbol{y}\} p_\mathcal{X}(\boldsymbol{x}), \tag{3}$$

and its prefix probability as in Eq. (1). Snæbjarnarson et al. (2026) give a computational shortcut for computing the prefix probability by defining the **quotient** and **remainder**. For $\boldsymbol{y} \in \mathcal{Y}^*$, we define the **precover** of $\boldsymbol{y}$ with respect to f as $\mathcal{P}(\boldsymbol{y}) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathcal{X}^* \colon \boldsymbol{y} \preceq f(\boldsymbol{x})\}$. Let

[1] App. A collects all notation in one place.

[2] An autoregressive model supplies these conditionals directly. They define a distribution over finite strings exactly when ancestral sampling reaches EOS with probability one (Cotterell et al., 2024).

$\mathcal{C}(\boldsymbol{y}) \stackrel{\text{def}}{=} \{\boldsymbol{x} \in \mathcal{X}^* : \langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})\}$, then $\mathcal{C}(\boldsymbol{y})$ is the largest union of cylinders $\langle \boldsymbol{x} \rangle$ contained in $\mathcal{P}(\boldsymbol{y})$. The quotient and remainder are $\mathcal{Q}(\boldsymbol{y}) \stackrel{\text{def}}{=} \text{pf}(\mathcal{C}(\boldsymbol{y}))$ and $\mathcal{R}(\boldsymbol{y}) \stackrel{\text{def}}{=} \mathcal{P}(\boldsymbol{y}) \setminus \mathcal{C}(\boldsymbol{y})$. Every completion of a quotient prefix covers $\boldsymbol{y}$, so we count their combined mass through its prefix probability. Remainder strings contribute individually, giving the **quotient–remainder decomposition**:

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}) = \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) + \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}). \tag{4}$$

Snæbjarnarson et al. (2026) show that the decomposition counts every source string exactly once, since the precover splits as the disjoint union $\mathcal{P}(\boldsymbol{y}) = \left(\bigcup_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \langle \boldsymbol{x} \rangle\right) \sqcup \mathcal{R}(\boldsymbol{y})$. Their algorithms use the three predicates below to determine how a source prefix $\boldsymbol{x}$ relates to the precover $\mathcal{P}(\boldsymbol{y})$:

Live, `is_live`$(\boldsymbol{x}, \boldsymbol{y})$: some extension covers $\boldsymbol{y}$, that is $\exists\, \boldsymbol{x}' \in \mathcal{X}^* : \boldsymbol{x}\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y})$.

Member, `is_member`$(\boldsymbol{x}, \boldsymbol{y})$: $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$, the output emitted so far covers $\boldsymbol{y}$.

Cylinder, `is_cylinder`$(\boldsymbol{x}, \boldsymbol{y})$: $\boldsymbol{x}\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y})$ for every $\boldsymbol{x}' \in \mathcal{X}^*$, every extension covers $\boldsymbol{y}$.

These predicates suffice to enumerate the quotient and remainder sets in Eq. (4). The first cylinder prefix on a path generates a cylinder of $\mathcal{C}(\boldsymbol{y})$ and so belongs to the quotient $\mathcal{Q}(\boldsymbol{y})$. A member that is not a cylinder is a remainder element of $\mathcal{R}(\boldsymbol{y})$. A non-live prefix is dropped. When f is encoded by a finite-state transducer, these can be evaluated in finite time.[3]

3 Unbiased Estimation of Transduced Language Models

The threshold-pruned beam summing of Snæbjarnarson et al. (2026) is biased and can be prohibitively expensive, as in DNA-to-amino-acid transduction. We replace its pruning function with resampling without replacement among the source prefixes produced at each extension step, avoiding repeated selections of the same prefix. We use *inclusion-probability* reweighting to preserve their weights in expectation (§3.2). We seek an estimate $\hat{Z}$ of $Z \stackrel{\text{def}}{=} \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$ that meets three criteria: (i) it is unbiased, i.e., $\mathbb{E}[\hat{Z}] = Z$, (ii) it has low variance, and (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 (Eq. (4)) is infinite, something the prior work cannot guarantee.

```
1  def beam_summing(y, ψ, prune):
2    N ← |y|
3    if N = 0:
4      (Q′, R′), Ẑ ← ({(ε, 1)}, ∅), []
5    else:
6      (Q′, R′), Ẑ ←
7        beam_summing(y_{<N}, ψ, prune)
8    P ← QUEUE(Q′)
9    (Q, R), A ← (∅, ∅), 0
10   for (x, w) ∈ R′:
11     if is_member(x, y):
12       R.add((x, w))
13       A += w
14   while |P| > 0:
15     P′ ← ∅
16     for x, w ∈ P:
17       if is_cylinder(x, y):
18         Q.add((x, w))
19         A += w
20         continue
21       if is_member(x, y):
22         R.add((x, w · p_X(EOS | x)))
23         A += w · p_X(EOS | x)
24       for x′ ∈ X with ψ(xx′) > 0:
25         P′.add((xx′, w · p_X(x′ | x)))
26     P ← prune(P′, A)
27   Ẑ.append(A)
28   return (Q, R), Ẑ
```

Figure 1: Beam summing for TLMs.

3.1 Beam Summing over the Precover Decomposition

The algorithm in Fig. 1 is the beam summing approach of Snæbjarnarson et al. (2026) made to return the estimand directly, here named `beam_summing`. To compute the quotient–remainder decomposition (Eq. (4)) for a nonempty target prefix, the algorithm first recurses on the prefix one position shorter. It collects quotient prefixes in `Q` (**quotient contributions**) and remainder strings in `R` (**remainder contributions**). We call the queue's entries **particles**, each a source prefix $\boldsymbol{x}$ having a nonnegative **weight** w, and the queue itself the **pool**. The queues and collections `Q` and `R` are multisets. With no pruning, a particle's weight is exactly its source prefix mass $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ (line 25). Within each call, `beam_summing` enumerates

[3] Following Snæbjarnarson et al. (2026), we implement these checks with lazy determinization. The algorithms otherwise require only these three predicates and apply to any representation that can compute them.

source prefixes breadth-first. The practical implementation of Snæbjarnarson et al. (2026) uses $\texttt{prune}_\tau$ to keep the enumeration manageable by deterministically keeping the heaviest particles accounting for at least $1-\tau$ of the mass, $0 \le \tau < 1$, biasing the estimate (Tab. 1, Fig. 13).[4]

After the recursive call, `beam_summing` initializes its pool from the returned quotient entries and passes the returned remainder entries through a membership re-check (line 11). A prefix with an entry $(\boldsymbol{x}, w) \in \mathsf{Q}'$ is a cylinder for $\boldsymbol{y}_{<N}$ but need not be one for $\boldsymbol{y}$, so the call may need to extend it further. 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 (lines 17 and 21). The pool is passed to `prune` after each iteration, and when it is empty, the call appends A to the returned list $\hat{Z}$. Snæbjarnarson et al. (2026) queue only source prefixes that pass the Live predicate. `beam_summing` generalizes this restriction through a **twist** ψ, a nonnegative weighting on source prefixes.[5] The **live twist** $\psi = \texttt{is\_live}$ has that restriction and is the twist our experiments use (§4). At target position t, both extension and pruning use a twist ψ_t that must be positive wherever the optimal twist ψ_t^* is positive. The pseudocode omits the target-position subscript. The extension step only spawns child particles with $\psi(\boldsymbol{x}x') > 0$ (line 24). The stochastic prune functions introduced below use the twisted weights $w\,\psi(\boldsymbol{x})$ to choose survivors (§3.2).

The optimal twist. Fix the target $\boldsymbol{y}$ and define the **optimal twist** of a source prefix as the probability that a completion (§2) covers the target,

$$\psi^*(\boldsymbol{x}) \stackrel{\text{def}}{=} \Pr_{X \sim p_{\mathcal{X}}}[\boldsymbol{y} \preceq f(X) \mid \boldsymbol{x} \preceq X] = \mathbb{E}_{\boldsymbol{x}'' \sim p_{\mathcal{X}}(\cdot \mid \boldsymbol{x})}[\mathbb{1}\{\texttt{is\_member}(\boldsymbol{x}\,\boldsymbol{x}'', \boldsymbol{y})\}]. \tag{5}$$

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 Eq. (11) under the conventions of §2). Computing ψ^* is as hard as the original problem, since $\psi^*(\varepsilon) = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$ (Lemma B.1). The live twist is positive wherever ψ^* is, as a twist must be, and is computable at every step. A look-ahead twist toward the complete target need not be valid for its prefixes, since it can zero children that carry mass for $\boldsymbol{y}_{\le t}$ but not for $\boldsymbol{y}$, so each position of the recursion needs its own valid twist which the live one satisfies (App. C.2).

3.2 Stochastic Pruning

Running `beam_summing` with either of the prune functions in Fig. 13 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 & 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 & Clifford, 2003). In `beam_summing`, particle i in the pool passed to the prune is a child particle $\boldsymbol{x}^{(m)}x'$ of its parent $\boldsymbol{x}^{(m)}$, with weight $w^{(i)} = w^{(m)}\,\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x}^{(m)})$ (Fig. 1). For a function $h^{(i)} \stackrel{\text{def}}{=} h(\boldsymbol{x}^{(i)})$, SWOR estimates $\sum_{i=1}^N w^{(i)}h^{(i)}$ by $\widehat{\Sigma}_h \stackrel{\text{def}}{=} \sum_{i\in\mathcal{S}} \frac{w^{(i)}}{\pi_i} h^{(i)}$. The estimate is unbiased for any valid inclusion probabilities, those with $\pi_i = 0$ only where $w^{(i)}h^{(i)} = 0$. With $I_i \stackrel{\text{def}}{=} \mathbb{1}\{i \in \mathcal{S}\}$ and $\mathbb{E}_{\mathcal{S}}[I_i] = \pi_i$,

$$\mathbb{E}_{\mathcal{S}}[\widehat{\Sigma}_h] = \sum_{i=1}^N \mathbb{E}_{\mathcal{S}}[I_i]\,\frac{w^{(i)}}{\pi_i}\,h^{(i)} = \sum_{i=1}^N w^{(i)}\,h^{(i)}. \tag{6}$$

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, we set each child particle's inclusion probability proportional to its weight, capped at one,

$$\pi_i = \min\bigl(1,\, c\,w^{(i)}\bigr), \quad i = 1, \dots, N, \qquad \sum_{i=1}^N \min\bigl(1,\, c\,w^{(i)}\bigr) = M, \tag{7}$$

[4] We leave the cap on the number of prefixes that Snæbjarnarson et al. (2026) use in high-entropy settings (e.g., DNA-to-amino-acid) out of the default baseline and report it separately in §4.2 and App. J.

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

Table 1: **The TLM algorithm family**. beam_summing (Fig. 1) forms every live child particle before pruning. beam_swor and beam_swor_ad (Fig. 2) extend beam_summing with pruning functions that sample distinct survivors without replacement. In contrast, smc_simple and smc_rb sample one child per particle. Only smc_simple samples the stop decision. The prune-free row is exact whenever the run halts. The stochastic methods include the halting mechanisms described in §3.4 and App. C.4. The variance labels summarize Tab. 2 and 5.

Algorithm	Prune	(i) Unbiased	(ii) Variance (qual.)	(iii) Particle count
beam_summing (Fig. 1)	none	✓	0	unbounded
↪ + prune_M	keep M heaviest	✗	—	fixed M
↪ + prune_τ	keep $\geq 1-\tau$ mass	✗	—	mass-set
smc_simple (App. C)	multinomial	✓	high	fixed M
smc_rb (App. C)	multinomial	✓	high	fixed M
beam_swor	SWOR	✓	low	fixed M
beam_swor_ad	SWOR, adaptive	✓	low	adaptive, $m \leq M$

```
def swor_{M,ψ}(P', A):
  W ← (w^(i) · ψ(x^(i)))_{i=1}^{|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 beam_swor(y, M, ψ):
  return beam_summing(y, ψ, swor_{M,ψ})

def beam_swor_ad(y, M, ψ):
  return
    beam_summing(y, ψ, swor_adaptive_{M,ψ,ρ})

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}
```

Figure 2: **SWOR.** The without-replacement resample swor replaces the prune in Fig. 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 lower bound in App. B.2. §3.4 gives the construction and its almost-sure halting guarantee. beam_swor and beam_swor_ad are beam_summing run with these prunes. Pseudocode for pps and systematic is given in Fig. 14.

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 & Kroese, 2018, §3.2.5), and every non-certain survivor has the same reweighted value $w^{(i)}/\pi_i$. This means that the reweighted total matches the pool total on every draw and that randomness only decides which prefixes carry the mass forward.

The pruning function swor (Fig. 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 given by Eq. (7) applied to the twisted weights, and draws the size-M subset with systematic (Fig. 14). 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 $w^{(i)}/\pi_i$. We write beam_swor for beam_summing run with the swor prune (Fig. 2).

3.3 Unbiasedness Guarantees

We now give sufficient conditions for when beam_summing (Fig. 1), combined with a prune function, gives an unbiased estimate. Without pruning, each particle carries its exact prefix mass (§3.1, Snæbjarnarson et al., 2026). For a pool $\mathcal{P}$, write $w_{\mathcal{P}}(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{(\boldsymbol{x}',w)\in\mathcal{P}} w\,\mathbb{1}\{\boldsymbol{x}' = \boldsymbol{x}\}$ for its weight at a source prefix $\boldsymbol{x} \in \mathcal{X}^*$. For every fixed choice of the inputs to the prune, let $\widetilde{\mathcal{P}} \sim \text{prune}(\mathcal{P}', \cdots)$ denote its random output. We require

$$\mathbb{E}_{\widetilde{\mathcal{P}}\sim\text{prune}(\mathcal{P}',\cdots)}[w_{\widetilde{\mathcal{P}}}(\boldsymbol{x})] = w_{\mathcal{P}'}(\boldsymbol{x}), \qquad \text{for every } \boldsymbol{x} \in \mathcal{X}^*. \tag{8}$$

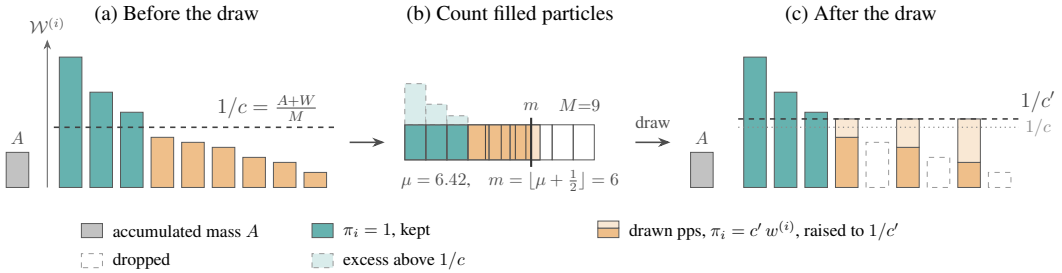


Figure 3: One `swor_adaptive` prune with the live twist and $\rho = 1$. (a) Colored bars show child-particle weights, and the gray bar shows accumulated mass A. The dashed line is $1/c = (A+W)/M$. (b) The capped counts sum to μ, fixing m survivors before sampling. (c) The draw uses a recomputed threshold $1/c'$. Particles at or above it keep their weights, other survivors get weight $1/c'$.

The expectation in Eq. (8) is only over the prune's internal randomness, with $\mathscr{P}'$ and A held fixed. Lemma B.2 shows the condition preserves the prune-free pool weights in expectation throughout the recursion. The following theorem establishes unbiasedness of the estimate (proof in App. B.1).

Theorem 3.1 (Unbiasedness)**.** *Fix a nonempty target string $\boldsymbol{y}$ and run* `beam_summing` *(Fig. 1) with a twist (§3.1). Suppose that (1) every prune preserves the pool's expected weight at every prefix (Eq. (8)), and (2) the run halts almost surely. Then $\mathbb{E}[\hat{Z}_{|\boldsymbol{y}|}] = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$.*

The draw `swor` meets hypothesis (1) by the Horvitz–Thompson identity (Eq. (6)), its inclusion probabilities are positive wherever the twisted weight is, and the extension step only hands it particles with positive twist (line 24). For hypothesis (2), the tail-roulette prune of App. C.4 ensures almost-sure halting while preserving hypothesis (1). For `beam_swor_ad`, both hypotheses follow from Proposition 3.2.

3.4 Adaptive-Budget SWOR and Almost-Sure Halting

For a finite quotient–remainder decomposition, every branch of the `beam_summing` enumeration reaches a cylinder or a dead end (a prefix with no live children) after finitely many iterations, so the call halts (Snæbjarnarson et al., 2026, Thm. 4.1). When the decomposition is infinite, limiting the particle count need not make the call halt (App. B.3).

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 Fig. 2, and `beam_swor_ad` is `beam_summing` run with it. We call this method **adaptive-budget SWOR**. At every `prune` call (line 49 of Fig. 2), we set

$$c = \frac{M}{\rho \cdot (A+W)}. \quad (9) \qquad \mu = \sum_{i=1}^{N} \min\Big(1,\ c w^{(i)} \psi(\boldsymbol{x}^{(i)})\Big), \qquad m = \min\big(M,\ \lfloor \mu + \tfrac{1}{2} \rfloor\big). \quad (10)$$

Here A is the current-position running total (Fig. 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 Fig. 3(a,b)). The resulting μ fixes the particle count m before sampling. This replaces the calculation of Eq. (7), where the constant is solved so that the capped counts sum to the fixed particle count M. `pps` then computes the inclusion probabilities for a size-m draw by re-solving the constant to c', and `systematic` selects the m survivors, so every non-pinned survivor carries the shared twisted weight $1/c'$ (Fig. 3(c)). In the limit $\rho \to 0$, `swor_adaptive` recovers the fixed-M `swor` prune. Under the live twist, the capped count satisfies $\mu \le (M/\rho)\, W/(A+W)$, 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 (line 52). To ensure halting even when no mass is accumulated, we use a positive lower bound for A in the particle-count calculation,

without introducing bias (App. B.2). The following proposition is proved in App. B.3. Note that almost-sure halting does not by itself guarantee finite expected runtime or finite estimator variance.

Proposition 3.2 (Adaptive-budget SWOR). *Fix a positive integer M, a twist ψ, and $\rho \in (0, 1]$, and run* `beam_swor_ad` *with the positive lower bound specified in App. B.2. Then (I) each prune call satisfies Eq. (8). Under the live twist $\psi =$* `is_live`*, (II) the full run halts almost surely. Consequently, under the live twist, $\mathbb{E}[\hat{Z}_t] = \overrightarrow{p_Y}(\boldsymbol{y}_{\leq t})$ at every position.*

3.5 SMC WITH MULTINOMIAL RESAMPLING

We compare our without-replacement methods with two SMC baselines using standard multinomial resampling with replacement (Tab. 1). `smc_simple` (Fig. 8) is a textbook particle filter (Doucet et al., 2001; Naesseth et al., 2019) run with the precover-restricted proposal, collecting mass when a particle draws EOS at a member prefix. `smc_rb` (Fig. 10) instead 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, whereas `beam_swor` forms every live child before pruning and samples distinct survivors without replacement. App. C develops both SMC baselines and their unbiasedness arguments.

4 EXPERIMENTS

We compare `smc_simple`, `smc_rb`, `beam_swor`, and `beam_swor_ad` with `beam_summing` using `prune`$_\tau$ (Tab. 1). Every setting uses the live twist `is_live`. Our implementation extends particles until they emit the current target prefix and estimates the remaining quotient and remainder contributions with sampled continuations, which keeps the estimates unbiased (App. E). For `beam_swor_ad`, we set ϵ to machine precision. For the other three stochastic methods, we apply the tail-roulette decorator (App. C.4). Prefix probabilities can span many orders of magnitude, so we report the log of the estimate averaged over S independent runs, $\log\left(\frac{1}{S}\sum_{s=1}^{S}\hat{Z}_s\right)$. Unlike averaging log estimates, this approaches the desired log probability as the number of runs grows, although logarithms and ratios of the estimates need not be unbiased. We compare against exact probabilities where available, and check the stability of the GPT-2 estimates and the effect of bias on the reading-time conclusions (Apps. I and K). We report between-seed standard deviations and mean live particle counts, with seed counts in Tab. 5.

4.1 BIAS AND VARIANCE

The Penn Treebank (PTB) tokenizer segments text into PTB units, e.g., splitting `!!!` into three successive `!` units (Marcus et al., 1993). Following Snæbjarnarson et al. (2026); Kiegeland et al. (2026), we write $\mathtt{f}_{\mathrm{ptb}}$ for this transducer and evaluate it on ten WikiText-2 paragraphs (Merity et al., 2017).

Bigram text 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 used by Snæbjarnarson et al. (2026).[6] We calculate the exact target prefix probabilities as described in App. G.

Fig. 4 plots the relative root-mean-square error across seeds[7]. Threshold-pruned beam summing and the with-replacement samplers have corpus error near 1, even when the samplers use thousands of particles. `beam_swor` and `beam_swor_ad` drive the corpus error down to 0.20 and 0.16 at their largest tested M values. Panel (c) shows they resolve most paragraphs where `smc_rb` needs M in the thousands and `smc_simple` never resolves the full corpus. App. K shows the comparison at every position.

GPT-2 text model. We run $\mathtt{f}_{\mathrm{ptb}}$ with GPT-2 Large, p_{gpt2} (Radford et al., 2019). Exact transduced prefix probabilities are unavailable for GPT-2, so we report the estimates and their between-seed variation as M grows and compare with the threshold-pruned lower bound (Fig. 19). The threshold-pruned log probability estimate lies below the `beam_swor_ad` estimate on all ten paragraphs, by 33.4 nats in total (Tab. 4). Resampling the 32 runs gives a 95% bootstrap interval of $[29.0, 34.7]$

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

[7] We use the relative $\mathrm{RMSE}[\hat{Z}/Z] = [S^{-1}\sum_{s=1}^{S}(\hat{Z}_s/Z - 1)^2]^{1/2}$.

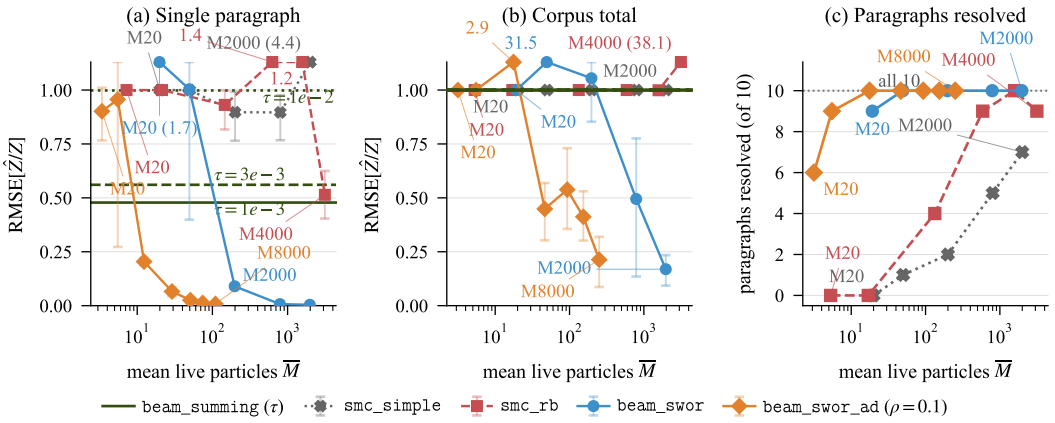


Figure 4: Unbiased methods versus beam summing with $\texttt{prune}_\tau$ on the $p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$ TLM, compared with exact probabilities. Panels (a) and (b) plot relative root-mean-square error $\mathrm{RMSE}[\hat{Z}/Z]$: (a) a single example paragraph and (b) the ten-paragraph corpus total, both against the mean live particle count. Labels give the maximum particle count M. Error bars give 95% bootstrap intervals. Points above the frame are drawn at its top with their value. Panel (c) counts the paragraphs each method resolves, capturing at least 5% of a paragraph's mass.

nats for the total. Because one paragraph's pooled estimate is dominated by a single run, we check its reproducibility with an independent batch and obtain a similar pooled log estimate (App. K).

DNA models. The DNA→amino-acid map f_{dna2aa} maps DNA strings over a four-letter alphabet to amino-acid strings. We train a bigram model on synthetic coding sequences (App. J), and use p_{dna}, a GPT-2 model trained on human DNA, the same model used by Snæbjarnarson et al. (2026). For both DNA models, relative RMSE against exact target prefix probabilities decreases faster with increasing particle count for `beam_swor_ad` than for `smc_rb` (App. J).

Variance comparison. 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. We also compare `beam_swor` with a variant that enumerates every live child but uses multinomial resampling. At the same maximum particle count, `beam_swor` gives lower relative RMSE on all ten paragraphs under the bigram model (App. K). `beam_swor_ad` matches `beam_swor` on GPT-2 and on DNA, where `swor_adaptive` retains the same number of particles as `swor`.

Table 2: Between-seed standard deviation of $\log \hat{Z}$ (nats), with seed counts in parentheses. Columns show the text corpus, first GPT-2 paragraph, and amino-acid target. `beam_swor_ad` uses ρ=0.1 and * marks M=8000

Algorithm M	$p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$ 2000, 8000*	$p_{\mathsf{gpt2}} \circ \mathsf{f}_{\mathrm{ptb}}$ 200	$p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{dna2aa}}$ 256
`smc_simple`	10.4 (32)	—	0.095 (32)
`smc_rb`	7.03 (30 of 32)	3.81 (32)	0.095 (32)
`beam_swor`	0.198 (32)	0.280 (8)	0.064 (32)
`beam_swor_ad`	0.161* (256)	0.282 (32)	0.064 (32)

4.2 Compute and Scaling

We compare the estimators against runtime in Figs. 5 and 20 (timing details in App. K). Under both source models, the `beam_swor_ad` frontier dominates both SMC baselines. For the bigram text model, `beam_swor_ad` at nominal M=8000 reaches a standard deviation of 0.16 nats in 623 seconds per seed, five times faster than `beam_swor` at M=2000, which reaches 0.20 nats in 3432 seconds.

The same ordering holds under the relative RMSE of the corpus estimate with zero estimates included (Tab. 5). At equal M, however, `beam_swor` has the lower relative RMSE. On $p_{\mathsf{gpt2}} \circ \mathsf{f}_{\mathrm{ptb}}$, `beam_swor_ad` at nominal M=200 takes 772 seconds, compared with 1224 seconds for $\texttt{beam\_summing}$ with $\texttt{prune}_\tau$. The M=800 point of Tab. 4 takes 1.5× as long as that baseline. The parameter ρ of `beam_swor_ad` is swept in App. F.

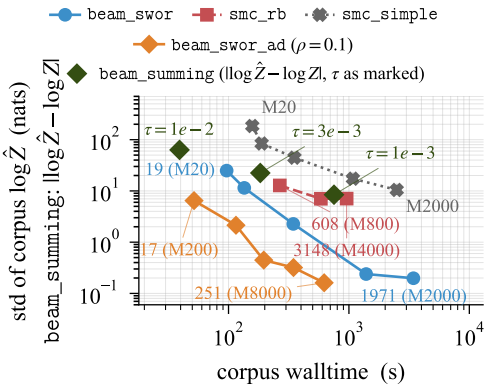


Figure 5: Between-seed std of the bigram corpus log-estimate against runtime per seed. For $\texttt{prune}_\tau$, we instead show absolute error against the exact log probability. Labels are mean live particles.

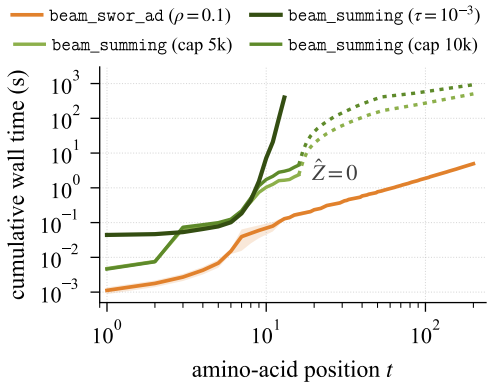


Figure 6: Runtime for $p_{\text{bigram}} \circ f_{\text{dna2aa}}$. `beam_swor_ad` uses M=256, ρ=0.1. Dotted segments mark positions where the capped `beam_summing` estimate is exactly zero.

Most amino acids can be encoded by several DNA codons, so a high-entropy source can spread mass over exponentially many source strings as the target grows. In the scaling comparison, `beam_summing` uses $\texttt{prune}_\tau$ with $\tau = 10^{-3}$. Fig. 6 shows the runtime of `beam_summing` with $\texttt{prune}_\tau$ growing rapidly within a dozen amino acids, while `beam_swor_ad` scales approximately linearly. With the additional candidate cap of Snæbjarnarson et al. (2026) at 5,000 or 10,000 candidates, the run finishes in about 500 or 930 seconds, but its estimate is exactly zero from position 17 onward. For `beam_swor_ad` at position 200, the log of the mean probability estimate over three runs is 0.059 nats above the exact value, computed with the bigram automaton of App. G. App. H 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 unbiased for any M.

4.3 Reading-Time Prediction

Surprisal theory hypothesizes that processing time increases with contextual surprisal, the negative log probability of a linguistic unit in context. Kiegeland et al. (2026) test this hypothesis on the MECO eye-tracking corpus (Siegelman et al., 2022), computing contextual surprisal from GPT-2 Small through transducers for PTB units, whitespace-delimited words, and characters. Each next-unit probability is estimated as $\hat{Z}_t/\hat{Z}_{t-1}$ using beam summing with $\texttt{prune}_\tau$ (Snæbjarnarson et al., 2026). Pruning lowers both estimates, but it can remove different fractions of their mass, so the ratio and resulting surprisal can err in either direction.To assess how this bias affects the conclusions, we rerun the analysis with PTB units using `swor_adaptive` in place of $\texttt{prune}_\tau$, otherwise retaining the experimental setup of Kiegeland et al. (2026) (App. I). We evaluate first-fixation duration, gaze duration, and total reading time.

Compared with $\texttt{prune}_\tau$, estimated corpus surprisal is 1.4 percent lower, a reduction of 106 nats (95% bootstrap interval $[102, 108]$ nats, App. I). We find no evidence of a predictive difference between the two fits on any reading-time measure (two-sided $p \geq 0.13$). Under either estimator, adding contextual surprisal computed over PTB units improves held-out prediction for all three reading-time measures (Tab. 3). Thus, replacing $\texttt{prune}_\tau$ with `swor_adaptive` leaves the qualitative conclusions unchanged.

5 Conclusion

This work replaces threshold pruning in the beam-summing method of Snæbjarnarson et al. (2026) with a weight-proportional draw without replacement and Horvitz–Thompson reweighting. `beam_swor_ad` uses the current-position running total to set its particle count. Its roulette rule ensures that the run halts almost surely. The resulting estimator is unbiased and replaces unknown pruning error with sampling variance, providing a reference for estimating threshold-pruning error. In the DNA→amino-acid setting, it also makes target prefix probability estimation feasible for longer targets over high-entropy sources.

AI USE STATEMENT

In this work, we used generative AI tools to assist with implementing and running parts of the experimental code, and give feedback on configurations and results. AI tools were also used to review and edit proofs. Additionally, we used generative AI tools to edit the paper for readability, structure, to create figures and help with LaTeXformating. Every AI-proposed edit was reviewed. We have not used generative AI tools to conceptualize the work. We have reviewed AI-assisted work by unit-testing against ground truth where possible, including a probabilistic tracer (Vieira, 2026; Shi et al., 2020). We take full responsibility for the content of this work, including text, claims or artifacts produced with the aid of generative AI.

ETHICS STATEMENT

This work uses only publicly available corpora and pretrained models, and the reading-time data come from the published MECO corpus without new data collection. We see no ethical concerns beyond those of language modeling in general.

REPRODUCIBILITY STATEMENT

The algorithms are given as pseudocode in Figs. 1 and 2, with the sampling helpers in App. D, and the SMC baselines in App. C. All proofs are in App. B. Every model, transducer, and evaluation set is named in §4, the pruning parameters and their selection are given in App. F, the exact reference probabilities are described in Apps. G and J, the reading-time reanalysis in App. I, and the hardware and timing protocol in App. K. The source code will be released publicly.

REFERENCES

Nicolas Chopin and Omiros Papaspiliopoulos. *An Introduction to Sequential Monte Carlo*. Springer Series in Statistics. Springer, 2020. doi: 10.1007/978-3-030-47845-2. URL https://doi.org/10.1007/978-3-030-47845-2.

Ryan Cotterell, Anej Svete, Clara Meister, Tianyu Liu, and Li Du. Formal aspects of language modeling, 2024. URL https://arxiv.org/abs/2311.04329.

Pierre Del Moral, Arnaud Doucet, and Ajay Jasra. Sequential Monte Carlo samplers. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 68(3):411–436, 2006. doi: 10.1111/j.1467-9868.2006.00553.x. URL https://doi.org/10.1111/j.1467-9868.2006.00553.x.

Arnaud Doucet, Nando de Freitas, and Neil J. Gordon (eds.). *Sequential Monte Carlo Methods in Practice*. Statistics for Engineering and Information Science. Springer, 2001. ISBN 978-1-4419-2887-0. doi: 10.1007/978-1-4757-3437-9. URL https://doi.org/10.1007/978-1-4757-3437-9.

Paul Fearnhead and Peter Clifford. On-line inference for hidden Markov models via particle filters. *Journal of the Royal Statistical Society: Series B (Statistical Methodology)*, 65(4):887–899, 2003. doi: 10.1111/1467-9868.00421. URL https://doi.org/10.1111/1467-9868.00421.

Jeremy Heng, Adrian N. Bishop, George Deligiannidis, and Arnaud Doucet. Controlled Sequential Monte Carlo. *The Annals of Statistics*, 48(5):2904–2929, 2020. doi: 10.1214/19-AOS1914. URL https://doi.org/10.1214/19-AOS1914.

D. G. Horvitz and D. J. Thompson. A generalization of sampling without replacement from a finite universe. *Journal of the American Statistical Association*, 47(260):663–685, 1952. doi: 10.1080/01621459.1952.10483446. URL http://www.jstor.org/stable/2280784.

Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. 3rd edition, 2026. URL https://web.stanford.edu/~jurafsky/slp3/. Online manuscript released August 19, 2026.

Herman Kahn. Use of different Monte Carlo sampling techniques. Paper P-766, RAND Corporation, Santa Monica, CA, 1955. URL https://www.rand.org/pubs/papers/P766.html.

Samuel Kiegeland, Vésteinn Snæbjarnarson, Tim Vieira, and Ryan Cotterell. 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)*, pp. 32202–32224, 2026. doi: 10.18653/v1/2026.acl-long.1485. URL https://aclanthology.org/2026.acl-long.1485/. Oral.

Wouter Kool, Herke Van Hoof, and Max Welling. Stochastic beams and where to find them: The Gumbel-top-k trick for sampling sequences without replacement. In Kamalika Chaudhuri and Ruslan Salakhutdinov (eds.), *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pp. 3499–3508. PMLR, 09–15 Jun 2019. URL https://proceedings.mlr.press/v97/kool19a.html.

Alexander K. Lew, Tan Zhi-Xuan, Gabriel Grand, and Vikash K. Mansinghka. Sequential Monte Carlo steering of large language models using probabilistic programs, 2023. URL https://arxiv.org/abs/2306.03081.

João Loula, 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. Syntactic and semantic control of large language models via sequential Monte Carlo. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=xoXn62FzD0.

William G. Madow. On the theory of systematic sampling, II. *The Annals of Mathematical Statistics*, 20(3):333–354, 1949. doi: 10.1214/aoms/1177729988. URL https://doi.org/10.1214/aoms/1177729988.

Mitchell P. Marcus, Beatrice Santorini, and Mary Ann Marcinkiewicz. Building a large annotated corpus of English: The Penn Treebank. *Computational Linguistics*, 1993. URL https://aclanthology.org/J93-2004/.

Clara Meister, Afra Amini, Tim Vieira, and Ryan Cotterell. Conditional Poisson stochastic beam search. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 664–681, 2021. doi: 10.18653/v1/2021.emnlp-main.52. URL https://aclanthology.org/2021.emnlp-main.52.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer Sentinel Mixture Models. In *International Conference on Learning Representations*, 2017. URL https://openreview.net/forum?id=Byj72udxe.

Mehryar Mohri. *Weighted Automata Algorithms*, pp. 213–254. Springer Berlin Heidelberg, 2009. ISBN 978-3-642-01492-5. doi: 10.1007/978-3-642-01492-5_6. URL https://doi.org/10.1007/978-3-642-01492-5_6.

Christian A. Naesseth, Fredrik Lindsten, and Thomas B. Schön. Elements of sequential Monte Carlo. *Foundations and Trends in Machine Learning*, 12(3):187–306, 2019. doi: 10.1561/2200000074. URL https://doi.org/10.1561/2200000074.

Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language Models are Unsupervised Multitask Learners. *OpenAI blog*, 1(8), 2019. URL https://cdn.openai.com/better-language-models/language_models_are_unsupervised_multitask_learners.pdf.

Rohan Shah and Dirk P. Kroese. Without-replacement sampling for particle methods on finite state spaces. *Statistics and Computing*, 28(3):633–652, May 2018. ISSN 1573-1375. doi: 10.1007/s11222-017-9752-8. URL https://doi.org/10.1007/s11222-017-9752-8.

Kensen Shi, David Bieber, and Charles Sutton. Incremental sampling without replacement for sequence models. In Hal Daumé III and Aarti Singh (eds.), *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 8785–8795. PMLR, 13–18 Jul 2020. URL https://proceedings.mlr.press/v119/shi20a.html.

Noam Siegelman, Sascha Schroeder, Cengiz Acartürk, Hee-Don Ahn, Svetlana Alexeeva, Simona Amenta, Raymond Bertram, Rolando Bonandrini, Marc Brysbaert, Daria Chernova, et al. Expanding horizons of cross-linguistic research on reading: The Multilingual Eye-movement Corpus (MECO). *Behavior Research Methods*, 54(6), 2022. URL https://link.springer.com/article/10.3758/s13428-021-01772-6.

Vésteinn Snæbjarnarson, Samuel Kiegeland, Tianyu Liu, Reda Boumasmoud, Tim Vieira, and Ryan Cotterell. Transducing language models. In *Proceedings of The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=qOyF214xmg.

Tim Vieira. Ideas for testing randomized algorithms. Blog post, Graduate Descent, 2026. URL https://timvieira.github.io/testing-randomized-algorithms/.

Tim Vieira, Benjamin Lebrun, Mario Giulianelli, Juan Luis Gastaldi, Brian Dusell, John Terilla, Timothy J. O'Donnell, and Ryan Cotterell. From language models over tokens to language models over characters. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu (eds.), *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 61391–61412. PMLR, 13–19 Jul 2025. URL https://proceedings.mlr.press/v267/vieira25a.html.

Nick Whiteley and Anthony Lee. Twisted particle filters. *The Annals of Statistics*, 42(1):115–141, 2014. doi: 10.1214/13-AOS1167. URL https://doi.org/10.1214/13-AOS1167.

Stephen Zhao, Rob Brekelmans, Alireza Makhzani, and Roger Grosse. Probabilistic Inference in Language Models via Twisted Sequential Monte Carlo. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24, pp. 60704–60748, 2024. URL https://proceedings.mlr.press/v235/zhao24c.html.

CONTENTS OF THE APPENDIX

A Notation Glossary **14**

B Deferred Proofs **15**

B.1 Proofs for the SWOR Estimator 15

B.2 Halting Without Accumulated Mass 17

B.3 Proof for Adaptive-Budget SWOR 17

C Sequential Monte Carlo over the Precover **20**

C.1 Sequential Monte Carlo 20

C.2 Intermediate Targets and the `smc_rb` Algorithm 21

C.3 Unbiasedness of `smc_rb` 23

C.4 A Common Tail-Roulette Prune 24

D Pruning Functions and Sampling Helpers **26**

E Implementation Details **27**

F Sensitivity to Pruning Parameters **27**

G Exact Ground Truth for Bigram Models **27**

H The Genetic-Code Worst Case **28**

I Details of the MECO Reanalysis **29**

J DNA-to-Amino-Acid Validation **30**

K Additional Byte-to-Word Results **31**

L Relation to Prior Work **34**

A Notation Glossary

Notation	Gloss
ε	Empty string
EOS	End-of-string symbol
$x, x' \in \mathcal{X}$	Symbols in the source alphabet $\mathcal{X}$
$\boldsymbol{x}, \boldsymbol{x}' \in \mathcal{X}^*$	Source strings
$\mathcal{X}^*$	Set of all source strings
$\boldsymbol{x}\boldsymbol{x}'$	Concatenation
$y \in \mathcal{Y}$	Symbols in the target alphabet $\mathcal{Y}$
$\boldsymbol{y} \in \mathcal{Y}^*$	Target strings, with $\boldsymbol{y}_{\leq t}$ the length-t target prefix
$\mathcal{Y}^*$	Set of all target strings
$f\colon \mathcal{X}^* \to \mathcal{Y}^*$	Functional string-to-string transformation
f	Transducer implementing f
$\boldsymbol{x} \preceq \boldsymbol{x}'$	$\boldsymbol{x}$ is a prefix of $\boldsymbol{x}'$
$\langle \boldsymbol{x} \rangle$	Cylinder spanned by $\boldsymbol{x}$, i.e. $\boldsymbol{x}\mathcal{X}^*$
$\mathrm{pf}(Z)$	Prefix-free operation on Z (§2)
$\mathcal{P}(\boldsymbol{y})$	Precover of $\boldsymbol{y}$: $\{\boldsymbol{x} : \boldsymbol{y} \preceq f(\boldsymbol{x})\}$ (Eq. (4))
$\mathcal{C}(\boldsymbol{y})$	Largest union of cylinders contained in $\mathcal{P}(\boldsymbol{y})$
$\mathcal{Q}(\boldsymbol{y})$	Quotient: prefix-free generators of $\mathcal{C}(\boldsymbol{y})$
$\mathcal{R}(\boldsymbol{y})$	Remainder: $\mathcal{P}(\boldsymbol{y}) \setminus \mathcal{C}(\boldsymbol{y})$
$p_{\mathcal{X}}$	Source language model over $\mathcal{X}^*$
$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$	Source prefix probability
$\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})$	Conditional next-symbol prefix probability
$\overrightarrow{p_{\mathcal{X}}}(\mathrm{EOS} \mid \boldsymbol{x})$	Source stop probability
$p_{\mathcal{Y}}$	Transduced (target) language model
$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$	Target prefix probability, the normalizing constant Z we estimate (Eq. (25))
$\texttt{is\_member}(\boldsymbol{x}, \boldsymbol{y})$	$\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y})$: the output emitted covers $\boldsymbol{y}$
$\texttt{is\_cylinder}(\boldsymbol{x}, \boldsymbol{y})$	$\langle \boldsymbol{x} \rangle \subseteq \mathcal{P}(\boldsymbol{y})$: every extension covers $\boldsymbol{y}$
$\texttt{is\_live}(\boldsymbol{x}, \boldsymbol{y})$	Some extension of $\boldsymbol{x}$ covers $\boldsymbol{y}$
$\mathcal{A}(\boldsymbol{x})$	Live set: live next source symbols $\{x' \in \mathcal{X} : \texttt{is\_live}(\boldsymbol{x}x', \boldsymbol{y})\}$
M	Fixed or maximum particle count
$\{(\boldsymbol{x}^{(m)}, w^{(m)})\}_{m=1}^{M}$	Particle pool: source prefixes and their weights
$w^{(m)}$	Particle weight
$w_0^{(m)}$	Reference weight inherited by particle m under tail roulette (App. C.4)
$\hat{Z}$	Prefix-probability estimate
$\psi^*(\boldsymbol{x})$, $\psi_t^*(\boldsymbol{x})$	Optimal twist of $\boldsymbol{x}$, relative to $\boldsymbol{y}$ or $\boldsymbol{y}_{\leq t}$
$\lambda_t(\boldsymbol{x})$	Twisted live mass, $\sum_{x' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x}) \psi_t(\boldsymbol{x}x')$
$\nu_t(\boldsymbol{x})$	Proposal normalizer, $\mathbb{1}\{\texttt{is\_member}(\boldsymbol{x}, \boldsymbol{y}_{\leq t})\}\, \overrightarrow{p_{\mathcal{X}}}(\mathrm{EOS} \mid \boldsymbol{x}) + \sum_{x'' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x'' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x'')$
ESS, η	Effective sample size (Eq. (29)), resample when ESS $< \eta M$
τ	Prior work's prune threshold, keep $\geq 1 - \tau$ of the pool's mass
A	Current-position running total, returned as $\hat{Z}_t$
ϵ	Halting constant
ρ	Scales $\texttt{swor\_adaptive}$'s threshold $1/c$
κ	Tail-roulette threshold relative to w_0 (App. C.4)
$\mathcal{K}$	Certainty set: child particles pinned to $\pi_i = 1$
$\mathcal{S}$	Sampled child set, of size M or m
$w^{(i)}/\pi_i$	Horvitz–Thompson reweighting

Notation conventions. **Color** encodes domain: magenta for source-domain objects ($\boldsymbol{x}$, $\mathcal{X}$, $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$) and olive for target-domain objects ($\boldsymbol{y}$, $\mathcal{Y}$, f). **Font** encodes type: lowercase italic for symbols (x, y), bold italic for strings ($\boldsymbol{x}$, $\boldsymbol{y}$), calligraphic for sets and alphabets ($\mathcal{X}$, $\mathcal{Y}$, $\mathcal{P}$, $\mathcal{Q}$, $\mathcal{R}$), and typewriter for the transducer (f) and for algorithm and predicate names ($\texttt{is\_member}$, $\texttt{is\_cylinder}$, $\texttt{is\_live}$, $\texttt{smc\_simple}$, $\texttt{resample}$). The overrightarrow $\overrightarrow{\cdot}$ marks prefix probabilities ($\overrightarrow{p_{\mathcal{X}}}$ is the prefix probability of $p_{\mathcal{X}}$).

B Deferred Proofs

This appendix gives the proofs deferred from the main text: the unbiasedness theorem for the SWOR estimator (App. B.1), the positive lower bound that ensures halting without accumulated mass (App. B.2), and the adaptive-budget proposition (App. B.3).

B.1 Proofs for the SWOR Estimator

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 B.1 (A backward recursion for the optimal twist)**.** *The optimal twist ψ^* of Eq. (5) (relative to $\boldsymbol{y}$) satisfies the backward recursion*

$$\psi^*(\boldsymbol{x}) = \begin{cases} 1 & \textit{if } \texttt{is\_cylinder}(\boldsymbol{x}, \boldsymbol{y}) \\ \mathbb{1}\{\texttt{is\_member}(\boldsymbol{x}, \boldsymbol{y})\}\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \\ \quad + \sum_{x' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \psi^*(\boldsymbol{x}x') & \textit{otherwise} \end{cases} \tag{11}$$

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

$$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\, \psi^*(\boldsymbol{x}) = \sum_{\substack{\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y}) \\ \boldsymbol{x} \preceq \boldsymbol{x}'}} p_{\mathcal{X}}(\boldsymbol{x}'), \tag{12}$$

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

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 Eq. (11).

When $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) = 0$, the recursion follows from the conventions of §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 Eq. (11) equal one. Otherwise, both sides equal $\mathbb{1}\{\texttt{is\_member}(\boldsymbol{x}, \boldsymbol{y})\}$.

It remains to prove Eq. (12). For $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) > 0$, the definition of the optimal twist gives

$$\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\, \psi^*(\boldsymbol{x}) = \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \sum_{\boldsymbol{x}'' \in \mathcal{X}^*} \frac{p_{\mathcal{X}}(\boldsymbol{x}\boldsymbol{x}'')}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})} \mathbb{1}\{\texttt{is\_member}(\boldsymbol{x}\boldsymbol{x}'', \boldsymbol{y})\} = \sum_{\substack{\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y}) \\ \boldsymbol{x} \preceq \boldsymbol{x}'}} p_{\mathcal{X}}(\boldsymbol{x}'). \tag{13}$$

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 Eq. (12) are zero.

Finally, setting $\boldsymbol{x} = \varepsilon$ in Eq. (12) gives

$$\psi^*(\varepsilon) = \sum_{\boldsymbol{x}' \in \mathcal{P}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}') = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}). \tag{14}$$

■

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 $\mathcal{P}_1$ 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 $\mathcal{P}_i'$ from $\mathcal{P}_i$, and the prune returns $\mathcal{P}_{i+1} = \texttt{prune}(\mathcal{P}_i', \cdots)$. Write $\overline{\mathcal{P}}_i$ 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 B.2 (Pruning preserves the unpruned pool weights in expectation)**.** *Fix a target string $\boldsymbol{y}$ and run* `beam_summing` *(Fig. 1), with some prune function satisfying Eq. (8). Then, at every iteration i and every source prefix $\boldsymbol{x}$,*

$$\mathbb{E}[w_{\mathscr{P}_i}(\boldsymbol{x})] = w_{\overline{\mathscr{P}}_i}(\boldsymbol{x}). \tag{15}$$

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

$$\mathbb{E}[w_{\mathscr{P}_1}(\boldsymbol{x})] = w_{\overline{\mathscr{P}}_1}(\boldsymbol{x}). \tag{16}$$

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

$$w_{\mathscr{P}'_i}(\boldsymbol{x}x') = w_{\mathscr{P}_i}(\boldsymbol{x})\, \mathbb{1}\{\neg\texttt{is\_cylinder}(\boldsymbol{x}, \boldsymbol{y})\}\, \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \mathbb{1}\{\psi(\boldsymbol{x}x') > 0\}. \tag{17}$$

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

$$\begin{aligned}
\mathbb{E}[w_{\mathscr{P}_{i+1}}(\boldsymbol{x}x')] &= \mathbb{E}[w_{\mathscr{P}'_i}(\boldsymbol{x}x')] && \text{(18a, condition on the prune inputs and apply Eq. (8))}\\
&= \mathbb{E}[w_{\mathscr{P}_i}(\boldsymbol{x})]\, \mathbb{1}\{\neg\texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y})\}\, \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \mathbb{1}\{\psi(\boldsymbol{x}x') > 0\} && \text{(18b, Eq. (17))}\\
&= w_{\overline{\mathscr{P}}_i}(\boldsymbol{x})\, \mathbb{1}\{\neg\texttt{is_cylinder}(\boldsymbol{x}, \boldsymbol{y})\}\, \overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \mathbb{1}\{\psi(\boldsymbol{x}x') > 0\} && \text{(18c, induction hypothesis)}\\
&= w_{\overline{\mathscr{P}}'_i}(\boldsymbol{x}x') && \text{(18d, Eq. (17) in the reference run)}\\
&= w_{\overline{\mathscr{P}}_{i+1}}(\boldsymbol{x}x'). && \text{(18e, } \overline{\mathscr{P}}_{i+1} = \overline{\mathscr{P}}'_i \text{ under the identity prune)}
\end{aligned}$$

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. ■

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

$$V_t(\boldsymbol{x}) \overset{\text{def}}{=} \sum_{(\boldsymbol{x}', w) \in \texttt{Q}_t} w\, \mathbb{1}\{\boldsymbol{x}' = \boldsymbol{x}\} \;+\; \sum_{(\boldsymbol{x}', w) \in \texttt{R}_t} \frac{w}{\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})}\, \mathbb{1}\{\boldsymbol{x}' = \boldsymbol{x}\}, \tag{19}$$

the per-prefix estimate of $\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ produced by the call for $\boldsymbol{y}_{\le t}$. 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

$$\hat{Z}_t \overset{\text{def}}{=} \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y}_{\le t})} V_t(\boldsymbol{x})\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \;+ \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}_{\le t})} V_t(\boldsymbol{x}), \tag{20}$$

the quotient–remainder decomposition Eq. (4) with each exact prefix mass replaced by V_t (Eq. (2)).

Theorem 3.1 (Unbiasedness)**.** *Fix a nonempty target string $\boldsymbol{y}$ and run* `beam_summing` *(Fig. 1) with a twist (§3.1). Suppose that (1) every prune preserves the pool's expected weight at every prefix (Eq. (8)), 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 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 Eq. (19). 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 (§3.1) therefore gives $\overline{V}_t(\boldsymbol{x}) = \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}_{\leq t})$, while $\overline{V}_t(\boldsymbol{x})\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) = p_{\mathcal{X}}(\boldsymbol{x})$ for $\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y}_{\leq t})$.

We first show that pruning preserves these contributions in expectation, $\mathbb{E}[V_t(\boldsymbol{x})] = \overline{V}_t(\boldsymbol{x})$ for every $1 \leq t \leq |\boldsymbol{y}|$ and every $\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y}_{\leq t}) \sqcup \mathcal{R}(\boldsymbol{y}_{\leq t})$, by induction on t. Fix such a position t and an element $\boldsymbol{x}$ of its decomposition. Since $\mathcal{P}(\boldsymbol{y}_{\leq t}) \subseteq \mathcal{P}(\boldsymbol{y}_{\leq t-1})$, the element also belongs to the preceding precover. The preceding quotient–remainder split gives two alternatives. Either $\boldsymbol{x}$ extends a unique $\boldsymbol{x}' \in \mathcal{Q}(\boldsymbol{y}_{\leq t-1})$, or $\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y}_{\leq t-1})$ (§2). 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 $\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) = 0$, the convention in Eq. (19) gives $V_t(\boldsymbol{x}) = \overline{V}_t(\boldsymbol{x}) = 0$. Thus, if $\boldsymbol{x}$ is a remainder element, only the case $\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) > 0$ 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, Eq. (19) 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 B.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 $\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y}_{\leq t})$ (line 11). 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

$$\begin{aligned}
\mathbb{E}[\hat{Z}_T] &= \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} \mathbb{E}[V_T(\boldsymbol{x})]\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \;+\; \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \mathbb{E}[V_T(\boldsymbol{x})] && \text{(21a, Eq. (20) and Tonelli)} \\
&= \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} \overline{V}_T(\boldsymbol{x})\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) \;+\; \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overline{V}_T(\boldsymbol{x}) && \text{(21b, the preservation above)} \\
&= \sum_{\boldsymbol{x} \in \mathcal{R}(\boldsymbol{y})} p_{\mathcal{X}}(\boldsymbol{x}) \;+\; \sum_{\boldsymbol{x} \in \mathcal{Q}(\boldsymbol{y})} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) && \text{(21c, the prune-free identities above)} \\
&= \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}). && \text{(21d, Eq. (4))}
\end{aligned}$$

■

B.2 Halting Without Accumulated Mass

To also cover a position that can continue forever without adding anything to A, we replace it at position t by

$$A_{\epsilon,t} \stackrel{\text{def}}{=} \max\{A, \epsilon \hat{Z}_{t-1}\}, \qquad \hat{Z}_0 \stackrel{\text{def}}{=} 1, \quad \epsilon > 0. \tag{22}$$

This substitution applies only to the count and threshold in line 49 and Eq. (9). 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.

B.3 Proof for Adaptive-Budget SWOR

Run beam_swor with the live twist, without tail roulette, on the one-state transducer of Fig. 7 and the target $\boldsymbol{y} = \mathsf{c}$. 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 is all quotient, $\mathcal{Q}(\mathsf{c}) = \{\mathsf{a}^k\mathsf{b} : k \geq 0\}$, one cylinder at every depth. 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, $\mathsf{a}^k\mathsf{b}$

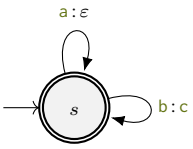


Figure 7: The one-state transducer of the halting example. Both arcs loop on the single initial, final state, a emitting nothing and b emitting c, so a source string maps to one c for each b it contains.

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 $M \geq 2$, the prune 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.

By Theorem 3.1, it is enough to show that every call to the adaptive-budget prune preserves the expected weight at each source prefix (Eq. (8)) and that the full run halts almost surely.

Proposition 3.2 (Adaptive-budget SWOR)**.** *Fix a positive integer M, a twist ψ, and $\rho \in (0, 1]$, and run* beam_swor_ad *with the positive lower bound specified in App. B.2. Then (I) each prune call satisfies Eq. (8). Under the live twist $\psi =$* is_live*, (II) the full run halts almost surely. Consequently, under the live twist, $\mathbb{E}[\hat{Z}_t] = \overrightarrow{p_Y}(\boldsymbol{y}_{\leq t})$ at every position.*

Proof. (I) To establish Eq. (8), 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 Fig. 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 $\mathscr{P}'$ (line 24). 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 Eq. (8).

(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 Eq. (8) 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 Eq. (6) gives

$$\mathbb{E}\left[w_{\widetilde{\mathscr{P}}}(\boldsymbol{x})\right] = w_{\mathscr{P}'}(\boldsymbol{x}),$$

which is Eq. (8).

(3) Suppose $W > 0$ and $m = 0$. Because $W > 0$, the definition of μ in Eq. (10) 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 Eq. (8), 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 Eq. (8) also holds in the third case.

Together, the three cases establish (I). Multiplying Eq. (8) 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_ad` 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,

$$A_{\epsilon,t} \geq \epsilon \hat{Z}_{t-1} > 0. \tag{23}$$

For $n \geq 1$, when the nth 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 nth 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 (§2), and the outer sum has finitely many terms.

At the nth prune, write $\mathcal{W}_n^{(i)}$ for its twisted particle weights, so $W_n = \sum_i \mathcal{W}_n^{(i)}$. Then Eq. (10) and (23) 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 Eq. (10) gives $m_n > 0$ exactly when $\mu_n \geq \frac{1}{2}$. Markov's inequality now gives

$$\Pr[m_n > 0] = \Pr\left[\mu_n \geq \tfrac{1}{2}\right] \leq 2\mathbb{E}[\mu_n] \xrightarrow{n\to\infty} 0. \tag{24}$$

Let E_n be the event that the call survives the nth 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

$$p_n \le \Pr[m_n > 0] + \tfrac{1}{2} p_{n-1}.$$

Taking $n \to \infty$ and using Eq. (24) gives $p \le \frac{1}{2}p$, so $p = 0$. The events E_n are decreasing, and their intersection is the event that the call never halts. Therefore

$$\Pr\left[\bigcap_{n \ge 1} E_n\right] = \lim_{n \to \infty} p_n = 0.$$

Thus the current call halts almost surely for every realized finite starting pool. The induction proves that the run halts almost surely. Applying Theorem 3.1 to each target prefix using (I) and (II) gives the final claim. ■

C Sequential Monte Carlo over the Precover

This appendix develops the with-replacement SMC algorithms for estimating the total mass of the precover, including the generic estimator and its resampling (App. C.1), the intermediate targets and the `smc_rb` algorithm (App. C.2), and the unbiasedness of `smc_rb` (App. C.3). Finally, it gives the common tail-roulette prune for the fixed-M methods (App. C.4).

C.1 Sequential Monte Carlo

The sum we wish to estimate (Eq. (1)) is over source strings filtered by a membership test in the precover $\mathcal{P}(\boldsymbol{y})$. A **potential** is a function $\phi\colon \mathcal{X}^* \to \mathbb{R}_{\ge 0}$ that weights each source string (Doucet et al., 2001), and Z is its expectation under the source model,

$$Z \stackrel{\text{def}}{=} \sum_{\boldsymbol{x} \in \mathcal{X}^*} p_{\mathcal{X}}(\boldsymbol{x}) \phi(\boldsymbol{x})\,. \tag{25}$$

For a transduced model, we take the potential to be the predicate `is_member` (§2), $\phi(\boldsymbol{x}) = \texttt{is\_member}(\boldsymbol{x}, \boldsymbol{y})$, for a fixed $\boldsymbol{y}$. The estimand Z is then the probability that a string drawn from the source model maps to a target string having $\boldsymbol{y}$ as a prefix, i.e., $Z = \overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y})$.

The algorithm estimating Z with SMC is shown in Fig. 8, and we now describe it. The pool $\mathcal{P}$ is initialized with M **particles** at the empty string, each with weight $1/M$ (line 62). At every iteration, a particle draws its next symbol by **sequential importance sampling** (Doucet et al., 2001), drawing from a **conditional proposal** distribution $\overrightarrow{q}(\cdot \mid \boldsymbol{x})$ (line 66) and correcting for it by multiplying its weight by the ratio of source to proposal conditionals (line 67),

$$w \leftarrow w \cdot \frac{\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})}{\overrightarrow{q}(x' \mid \boldsymbol{x})}. \tag{26}$$

A particle that draws EOS is **completed**: it leaves the pool, and its weight, multiplied by the potential, is added to the estimate $\hat{Z}$ (lines 69 and 73). Assume for a moment that we do not call `resample`. Then, for a single particle, the factors of Eq. (26), the potential, and the initial $1/M$ multiply to $\frac{1}{M} w(\boldsymbol{x})$, with the **importance weight**

$$w(\boldsymbol{x}) \stackrel{\text{def}}{=} \phi(\boldsymbol{x}) \frac{\overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})}{\overrightarrow{q}(\text{EOS} \mid \boldsymbol{x})} \prod_{s=1}^{|\boldsymbol{x}|} \frac{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}_s \mid \boldsymbol{x}_{<s})}{\overrightarrow{q}(\boldsymbol{x}_s \mid \boldsymbol{x}_{<s})} = \frac{\phi(\boldsymbol{x}) p_{\mathcal{X}}(\boldsymbol{x})}{q(\boldsymbol{x})}. \tag{27}$$

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

$$\phi(\boldsymbol{x})\, p_{\mathcal{X}}(\boldsymbol{x}) > 0 \implies q(\boldsymbol{x}) > 0 \qquad \text{for every } \boldsymbol{x} \in \mathcal{X}^*, \tag{28}$$

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 Eq. (27).

```
60 def smc_simple(q→, φ, M):
61   R ← ∅
62   P ← {(ε, 1/M)}_{m=1}^M
63   while |P| > 0:
64     P' ← ∅
65     for (x, w) ∈ P:
66       sample x' ~ q→(· | x)
67       w ← w · p→_X(x' | x) / q→(x' | x)
68       if x' = EOS:
69         R.add((x, w · φ(x)))
70       else:
71         P'.add((xx', w))
72     P ← resample(P')
73   return Σ_{(x,w)∈R} w

74 def resample(P):
75   if |P| = 0: return ∅
76   if Σ_i w^(i) = 0: return ∅
77   if ESS(P) ≥ η|P|:
78     return P
79   j_1, ..., j_|P| ~(i.i.d.) Categorical(w^(i) / Σ_i w^(i))
80   return {(x^(j_k), Σ_i w^(i)/|P|)}_{k=1}^|P|
```

Figure 8: A minimal **sequential Monte Carlo** algorithm for estimating Z (Eq. (25)). M particles extend one symbol at a time under the proposal, with the incremental weights of Eq. (26), finish with the potential factor at EOS, and hand uneven weights to `resample`. `resample` is an ESS-gated multinomial resample (Eq. (29)) that samples with replacement and resets weights to their mean.

Without resampling, the particle weights can become increasingly unequal as source prefixes grow, until a single particle dominates (Doucet et al., 2001). The **effective sample size**

$$\mathrm{ESS} = \frac{\left(\sum_{m=1}^{|\mathcal{P}|} w^{(m)}\right)^2}{\sum_{m=1}^{|\mathcal{P}|} \left(w^{(m)}\right)^2} \tag{29}$$

quantifies that imbalance. When it falls below $\eta|\mathcal{P}|$ (line 77), for a fixed $0 < \eta < 1$, `resample` redraws the pool with replacement proportionally to weight and resets every weight to the pool mean (lines 79 and 80), so the higher-weighted particles are duplicated and the lower ones discarded. When the support condition Eq. (28) 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.

C.2 Intermediate Targets and the `smc_rb` Algorithm

In App. C.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 (Fig. 11). 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**

$$\mu_t(\boldsymbol{x}) \stackrel{\text{def}}{=} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}) \cdot \psi_t(\boldsymbol{x}). \tag{30}$$

As in §3.1, 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 et al., 2006; Naesseth et al., 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 (App. B.1), taken here relative to $\boldsymbol{y}_{\leq t}$, and define the **twisted live mass**

$$\lambda_t(\boldsymbol{x}) \stackrel{\text{def}}{=} \sum_{x'' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x'' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x''). \tag{31}$$

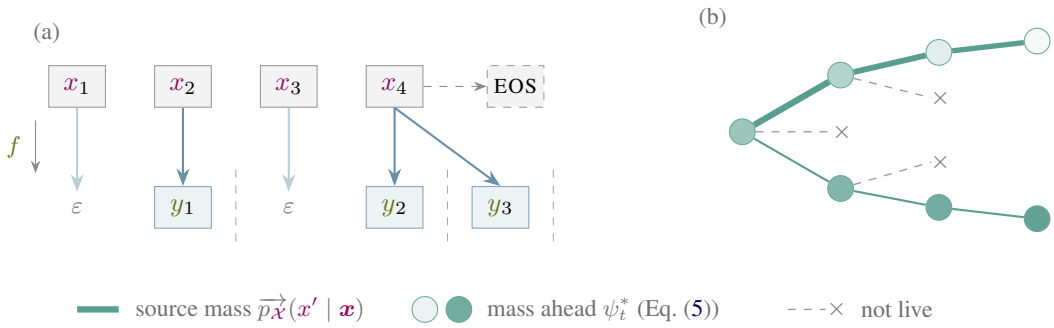


Figure 9: (a) An example source string aligned to a target prefix under a function f, the dashed lines marking the boundaries between target positions. A symbol may emit none or several target symbols, so the source spans between boundaries vary in length while the alignment stays monotone. At each step a *source* symbol is scanned, and at each boundary a *target* symbol is emitted. (b) When $\psi = \texttt{is\_live}$ we sample by source mass (edge width) and never enter a dead branch ($\times$), so no draw is wasted. Since we only look at the current live symbols, we cannot tell a heavy branch with little mass ahead (top) from a light branch that carries it (bottom). Importance weighting corrects for the proposal, while resampling with replacement mitigates particle-weight degeneracy (Apps. C.1 and C.3).

The proposal distribution draws the next symbol with probability proportional to $\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x')$,

$$\overrightarrow{q_t}(x' \mid \boldsymbol{x}) = \frac{\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x')}{\nu_t(\boldsymbol{x})}, \qquad \overrightarrow{q_t}(\text{EOS} \mid \boldsymbol{x}) = \frac{\mathbb{1}\{\texttt{is\_member}(\boldsymbol{x}, \boldsymbol{y}_{\leq t})\}\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x})}{\nu_t(\boldsymbol{x})},$$
$$\nu_t(\boldsymbol{x}) \stackrel{\text{def}}{=} \mathbb{1}\{\texttt{is\_member}(\boldsymbol{x}, \boldsymbol{y}_{\leq t})\}\, \overrightarrow{p_{\mathcal{X}}}(\text{EOS} \mid \boldsymbol{x}) + \sum_{x'' \in \mathcal{A}(\boldsymbol{x})} \overrightarrow{p_{\mathcal{X}}}(x'' \mid \boldsymbol{x})\, \psi_t(\boldsymbol{x}x''), \tag{32}$$

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 (Fig. 9b). We accept the compromise because the live predicate is computable at every step (§2), whereas the full mass ahead is the optimal twist of Eq. (5), exactly the quantity we cannot compute. A look-ahead twist must remain positive wherever ψ_t^* is positive at each target position t (§3.1). Ruling out a source prefix because it cannot cover the complete target $\boldsymbol{y}$ can discard mass needed for an earlier target prefix $\boldsymbol{y}_{\leq t}$. Richer twists, including learned ones, can be combined with our pruning functions subject to this validity condition. Designing such twists is a separate problem from the unbiased pruning and recursive estimation studied here. We use the live twist throughout and leave richer twists to future work.

Importance sampling in the target recursion. `smc_rb` iterates over the target and source indices as shown in the example Fig. 9a. A **target step** occurs once when the recursion advances from target position $t-1$ to t. At a fixed source prefix, the weight becomes

$$w \;\leftarrow\; w \cdot \frac{\mu_t(\boldsymbol{x})}{\mu_{t-1}(\boldsymbol{x})} = w \cdot \frac{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\psi_t(\boldsymbol{x})}{\overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x})\psi_{t-1}(\boldsymbol{x})} = w \cdot \frac{\psi_t(\boldsymbol{x})}{\psi_{t-1}(\boldsymbol{x})}. \tag{33}$$

The **source step** advances by drawing one symbol from the twisted proposal. Its incremental importance weight is

$$\frac{\mu_t(\boldsymbol{x}x')/\mu_t(\boldsymbol{x})}{\overrightarrow{q_t}(x' \mid \boldsymbol{x})} = \frac{\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\psi_t(\boldsymbol{x}x')}{\psi_t(\boldsymbol{x})} \cdot \frac{\nu_t(\boldsymbol{x})}{\overrightarrow{p_{\mathcal{X}}}(x' \mid \boldsymbol{x})\psi_t(\boldsymbol{x}x')} = \frac{\nu_t(\boldsymbol{x})}{\psi_t(\boldsymbol{x})}. \tag{34}$$

Therefore,

$$x' \sim \overrightarrow{q_t}(\cdot \mid \boldsymbol{x}), \qquad (\boldsymbol{x}, w) \leftarrow \left(\boldsymbol{x}x', w\frac{\nu_t(\boldsymbol{x})}{\psi_t(\boldsymbol{x})}\right), \tag{35}$$

with $\nu_t(\boldsymbol{x})$ the proposal's normalizer of Eq. (32). 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

$$w \;\leftarrow\; w \cdot \frac{\mu_t(\boldsymbol{x}_{\le n+k})}{\mu_{t-1}(\boldsymbol{x}_{\le n}) \prod_{j=1}^{k} \overrightarrow{q_t}(\boldsymbol{x}_{n+j} \mid \boldsymbol{x}_{<n+j})} = w \cdot \frac{\psi_t(\boldsymbol{x}_{\le n})}{\psi_{t-1}(\boldsymbol{x}_{\le n})} \prod_{j=1}^{k} \frac{\nu_t(\boldsymbol{x}_{<n+j})}{\psi_t(\boldsymbol{x}_{<n+j})}, \tag{36}$$

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 Eq. (32) on drawing a live next source symbol. Substituting the conditional proposal into Eq. (26), the update simplifies to

$$w^{(m)} \leftarrow w^{(m)} \cdot \lambda_t(\boldsymbol{x}^{(m)}). \tag{37}$$

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 (Fig. 10).

C.3 Unbiasedness of `smc_rb`

We now show that `smc_rb` is unbiased under the live twist. The proof follows App. B.1 for `beam_summing` but with the sampled symbol extension instead of all live extensions.

Proposition C.1 (Unbiasedness of `smc_rb`). *Fix a nonempty target string $\boldsymbol{y}$ and run `smc_rb` (Fig. 10) with the live twist $\psi =$ `is_live`. Suppose that every prune preserves the pool's expected weight at every prefix (Eq. (8)) 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 B.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 Eq. (17).

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 Eq. (32), conditioned on a live source symbol. For any next source symbol x' with $\psi(\boldsymbol{x}x') > 0$, its conditional expected contribution is

$$\mathbb{E}_X\left[\mathbb{1}\{X = x'\}\frac{w\,\lambda(\boldsymbol{x})}{\psi(\boldsymbol{x}x')}\right] = \frac{\overrightarrow{p_X}(x' \mid \boldsymbol{x})\psi(\boldsymbol{x}x')}{\lambda(\boldsymbol{x})}\frac{w\,\lambda(\boldsymbol{x})}{\psi(\boldsymbol{x}x')} = w\,\overrightarrow{p_X}(x' \mid \boldsymbol{x}). \tag{38}$$

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 Eq. (38) over the particles at $\boldsymbol{x}$ gives the conditional-expectation version of Eq. (17). The tower rule and Eq. (8) then close the same within-call induction and give, for every iteration i and source prefix $\boldsymbol{x}$,

$$\mathbb{E}[w_{\mathscr{P}_i}(\boldsymbol{x})] = w_{\overline{\mathscr{P}}_i}(\boldsymbol{x}). \tag{39}$$

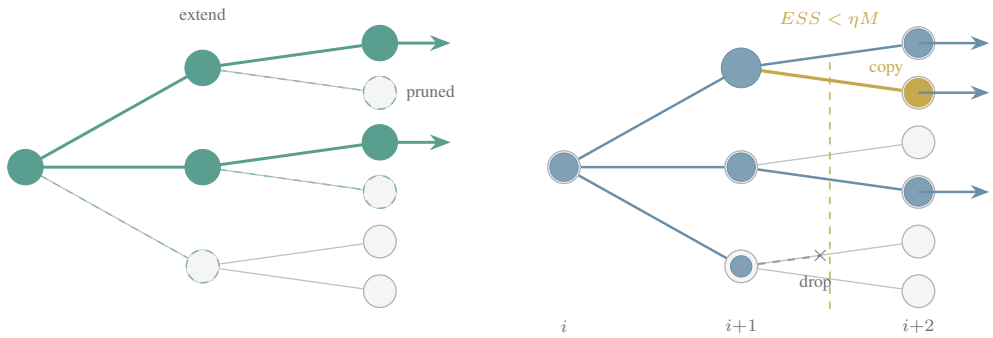
Marginalizing the stop decision replaces its sampled contribution by its conditional expectation. At a member, the baseline proposal stops with probability $\overrightarrow{p_X}(\text{EOS} \mid \boldsymbol{x})/\nu_t(\boldsymbol{x})$ and returns weight $w\nu_t(\boldsymbol{x})$, so

$$\frac{\overrightarrow{p_X}(\text{EOS} \mid \boldsymbol{x})}{\nu_t(\boldsymbol{x})} w\,\nu_t(\boldsymbol{x}) = w\,\overrightarrow{p_X}(\text{EOS} \mid \boldsymbol{x}), \tag{40}$$

```
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 · p⃗_X(EOS | x)))
        A += w · p⃗_X(EOS | x)
      λ ← Σ_{x′∈X} p⃗_X(x′ | x) · ψ(xx′)
      if λ = 0:
        continue
      sample x′ ~ p⃗_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′, A):
  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}
```

Figure 10: The smc_rb algorithm and ESS-gated multinomial resampling under a twist.



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

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

Figure 11: 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. Grey edges show unexplored branches. Dot size shows weight.

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 3.1 therefore applies to the pool identity above, and Eq. (21a, Eq. (20) and Tonelli) and (21d, Eq. (4)) give the result. ■

The ESS-gated multinomial resample of Fig. 10 satisfies Eq. (8) for every fixed input pool (Chopin & Papaspiliopoulos, 2020, Prop. 16.3).

C.4 A Common Tail-Roulette Prune

Source-prefix enumeration can continue indefinitely even as the remaining particle weights tend to zero (App. B.3). The decorator in Fig. 12 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 an original pruning function that returns at most M particles. We first show that the decorator preserves the particle weight in expectation whenever the original pruning function does. We then show that the algorithms with a fixed particle cap M in Tab. 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 Eq. (8) whenever the original does.

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

$$W_i \stackrel{\text{def}}{=} \sum_{(\boldsymbol{x}^{(m)}, w^{(m)}, w_0^{(m)}) \in \mathscr{P}_i^\circ} w^{(m)}. \tag{41}$$

By Eq. (8) and Lemma B.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 Eq. (38) gives the same result for `smc_rb`, and the importance-weight identities Eq. (26) and (27) give it for `smc_simple`. For $\texttt{prune}_M$, the expected total is no larger because its prune only removes particles. In every case,

$$\mathbb{E}[W_i] \le \sum_{m=1}^{|\mathscr{P}_1|} w_0^{(m)} \sum_{\boldsymbol{x}' \in \mathcal{X}^i} \overrightarrow{p_{\mathcal{X}}}(\boldsymbol{x}' \mid \boldsymbol{x}^{(m)}) \xrightarrow{i \to \infty} 0. \tag{42}$$

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 $i \to \infty$ (§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

$$\delta_{\min} \stackrel{\text{def}}{=} \kappa \min_{1 \le m \le |\mathscr{P}_1|} w_0^{(m)} > 0. \tag{43}$$

Because descendants inherit w_0, every descendant's coin-flip threshold κw_0 is at least $\delta_{\min}$.

For $i \ge 1$, 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

$$\Pr[W_i \ge \delta_{\min}] \le \frac{\mathbb{E}[W_i]}{\delta_{\min}} \longrightarrow 0, \tag{44}$$

$$\Pr\left[\widetilde{\mathscr{P}}_i \ne \emptyset\right] \le \Pr[W_i \ge \delta_{\min}] + (1 - 2^{-M}) \Pr\left[\widetilde{\mathscr{P}}_{i-1} \ne \emptyset\right], \qquad i \ge 2. \tag{45}$$

Let $E_i \stackrel{\text{def}}{=} \{\widetilde{\mathscr{P}}_i \ne \emptyset\}$ 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, $(p_i)_{i \ge 1}$ is nonincreasing and bounded below by zero, so it converges to some $p \ge 0$. Taking $i \to \infty$ in the recursion, with $p_i \to p$, $p_{i-1} \to p$, and $\Pr[W_i \ge \delta_{\min}] \to 0$, gives $p \le (1 - 2^{-M})p$. Since $1 - 2^{-M} < 1$, this forces $p = 0$. The intersection $\bigcap_{i \ge 1} E_i$ is the event that the pool remains nonempty at every iteration. For decreasing events, its probability is the limit of their probabilities, so $\Pr\left[\bigcap_{i \ge 1} E_i\right] = \lim_{i \to \infty} p_i = 0$. 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.

```
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
```

Figure 12: Tail-roulette decorator for a pruning function.

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_ad` does not require the decorator because its adaptive prune uses $A_{\epsilon,t}$ to ensure almost-sure halting (Proposition 3.2).

D Pruning Functions and Sampling Helpers

This appendix gives the two deterministic pruning functions of the prior work, $\texttt{prune}_M$ and $\texttt{prune}_\tau$ (Fig. 13), and the two sampling helpers that `swor` and `swor_adaptive` call (Fig. 14).

```
def prune_M(𝒫′, A):
  # 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′, A):
  # 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}
```

Figure 13: `beam_summing` with $\texttt{prune}_M$, keeping the M heaviest particles, is standard beam search, and $\texttt{prune}_\tau$ keeps the heaviest particles holding at least $1 - \tau$ of the pool's mass.

```
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}
```

Figure 14: Sampling helpers for Fig. 2. `pps` computes the capped inclusion probabilities of Eq. (7), and `systematic` draws a size-M subset with those probabilities.

Numerical implementation. We store particle weights and prefix-probability estimates in log space. For pruning, we rescale the pool weights and accumulated mass by a common factor and compute in double precision. The retained weights are then converted back to log space.

E Implementation Details

Following Snæbjarnarson et al. (2026), we represent each source prefix by the transducer states it can reach together with its unmatched output, and evaluate the predicates of §2 on this representation. Within the call for $\boldsymbol{y}_{\leq t}$, particles are extended and pruned until each has emitted $\boldsymbol{y}_{\leq t}$, and the particles already at this boundary count toward the particle budget M. The boundary particles form a prefix-free set, so by Eq. (12) the target prefix probability equals the sum of their prefix masses times their optimal twists ψ_t^*. We therefore estimate $\overrightarrow{p_Y}(\boldsymbol{y}_{\leq t})$ by $\sum_i w^{(i)}\hat{\psi}^{(i)}$, where $\hat{\psi}^{(i)}$ is an unbiased estimate of $\psi_t^*(\boldsymbol{x}^{(i)})$ obtained by sampling source continuations from the live proposal and accumulating the quotient and remainder contributions they reach. Conditional on the pool, the continuations are independent of the pruning, so Lemma B.2 and the tower rule give $\mathbb{E}[\hat{Z}_t] = \overrightarrow{p_Y}(\boldsymbol{y}_{\leq t})$. We draw one continuation per particle and pool the draws of particles with the same source prefix, which share the same ψ_t^*, with at most 512 continuations per distinct prefix. `smc_rb`, `beam_swor`, `beam_swor_ad`, and the all-child ablation share this estimator, `smc_simple` samples each continuation to EOS directly, and all reported runtimes include the continuations. The tail roulette of App. C.4 is applied along these continuations and keeps each estimate unbiased.

F Sensitivity to Pruning Parameters

We set ϵ to machine precision, 2^{-52}. Besides M, the adaptive-budget prune uses ρ (§3.4), and the ESS-gated multinomial prune uses η (App. C). We vary both on the ten WikiText paragraphs of §4.1, using the bigram model and eight seeds (Fig. 15). We selected the default $\rho = 0.1$ in a preliminary comparison on the same paragraphs, before the sweep reported here. We held it fixed for the GPT-2, DNA, and MECO experiments.

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 over the sweep's eight seeds, at 623 seconds per seed. Plain `beam_swor` at M=2000 gives 0.170 nats over the same eight seeds, at 3432 seconds per seed. Using all available seeds at these settings (256 for `beam_swor_ad` and 32 for `beam_swor`) gives standard deviations of 0.161 and 0.198 nats, respectively. `beam_swor_ad` therefore reaches a lower standard deviation at about one fifth of the time per seed (Tab. 5). 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 runtime 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.

G Exact Ground Truth for Bigram Models

The bigram model can be represented as a weighted finite-state acceptor. Composing it with $\mathsf{f}_{\mathrm{ptb}}$ and projecting onto the target labels gives a weighted automaton defining $p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$. In the representation used here, the functional transducer $\mathsf{f}_{\mathrm{ptb}}$ 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)

$$\beta = f + T\beta, \qquad \text{so} \qquad \beta = (I + T + T^2 + \cdots)f = (I - T)^{-1}f. \tag{46}$$

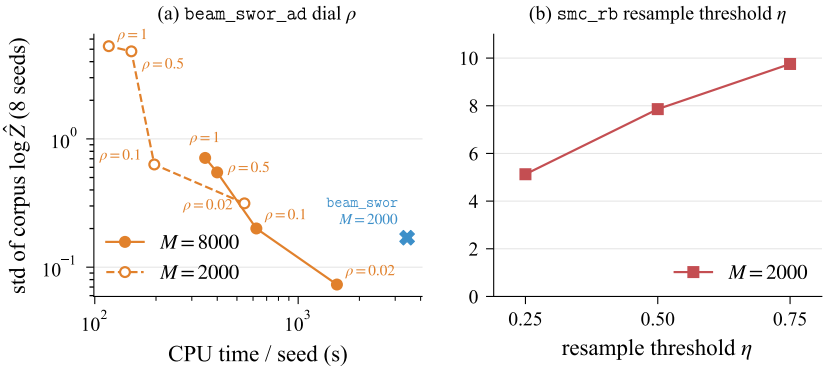


Figure 15: Sensitivity to the pruning parameters on the ten-paragraph bigram corpus (eight seeds). (a) Between-seed standard deviation of the corpus $\log \hat{Z}$ against runtime per seed as ρ varies for `beam_swor_ad` at nominal $M \in \{2000, 8000\}$. Labels give ρ, and plain `beam_swor` at $M{=}2000$ 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 $M{=}200$.

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

$$\overrightarrow{p_{\mathcal{Y}}}(\boldsymbol{y}_{\leq t}) = \sum_{\boldsymbol{x} \in \mathcal{P}(\boldsymbol{y}_{\leq t})} p_{\mathcal{X}}(\boldsymbol{x}) = \sum_{s \in \mathsf{S}} \alpha_t(s)\beta(s) = \alpha_t^{\top}\beta. \tag{47}$$

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.

H The Genetic-Code Worst Case

This appendix analyzes a worst case for the DNA setting in §4.2. Standard beam search captures an exponentially vanishing share of the target mass, while `smc_simple`, `smc_rb`, and fixed-M `beam_swor` are unbiased for any M. 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 `prune`$_M$ retains at most M of these d^T source strings, yielding

$$\hat{Z}_{\text{beam}} \leq M \cdot \left(\frac{1}{4}\right)^{3T}, \qquad \frac{\hat{Z}_{\text{beam}}}{p_{\mathcal{Y}}(\boldsymbol{y})} \leq \frac{M}{d^T}. \tag{48}$$

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 (Eq. (26) and (37)). After T amino acids every particle carries

$$w^{(m)} = \frac{1}{M}\frac{d^T}{4^{3T}} = \frac{p_{\mathcal{Y}}(\boldsymbol{y})}{M}, \tag{49}$$

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.

I Details of the MECO Reanalysis

We compare contextual surprisals computed by beam summing with prune$_\tau$ and swor_adaptive on the same 2,264 PTB units in the 12 MECO trials (individually read paragraphs). We evaluate first-fixation duration, gaze duration, and total reading time. These are, respectively, the duration of the first fixation on a word, the sum of its first-pass fixations, and the sum of all its fixations including refixations.

We use the generalized additive mixed models (GAMMs) of Kiegeland et al. (2026), which model reading times with a log-normal distribution. The baseline includes unit length and unigram surprisal for the current unit and the two preceding units. The target model adds contextual surprisal at the same three positions. Both models include a participant random intercept and by-participant random slopes for every predictor. For each measure, ΔLL is the per-observation improvement in held-out log likelihood from adding contextual surprisal to the shared baseline. We use 12-fold leave-one-trial-out cross-validation and report per-observation held-out ΔLL. Confidence intervals come from a 1,000-resample trial-level bootstrap, and significance is assessed with one-sided paired sign-flip tests. We change only the pruning function used to compute contextual surprisal and rerun the fits on the same observations (n=33,472). The prune$_\tau$ values are the telescoping estimates reported by Kiegeland et al. (2026), 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 120 estimation runs took 14.1 GPU-hours in total, and each GAMM fit takes about 20 seconds. 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 §4.3 to approximately 107 nats. Refitting with the corrected surprisals moves every held-out ΔLL by at most 0.015×10^{-3} nats and changes neither the results in Tab. 3 nor the between-estimator comparison below.

Size and location of the shift. Corpus surprisal is 7,486 nats under prune$_\tau$ and 7,380 nats under swor_adaptive. The median change per unit is approximately zero, and surprisal is higher under

Table 3: Predictive contribution of contextual surprisal over PTB units on MECO. Values are ΔLL in 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^{***}

swor_adaptive on 51 percent of the units. The 23 units with the largest reductions, one percent of the corpus, account for 95 percent of the total reduction. Twelve units have reductions greater than one nat, with a maximum of 14.5 nats.

Monte Carlo uncertainty. We draw 4,000 bootstrap samples by resampling the ten runs within each trial with replacement. For each sample, we average the prefix-probability estimates at every position before recomputing unit surprisals and their corpus total. The resulting 95% interval for the 106-nat reduction is $[101.7, 107.6]$ nats, with a bootstrap standard deviation of 1.5 nats. Leaving out any single run changes the corpus reduction by at most one nat and an individual unit's surprisal by at most 0.28 nats.

Between-estimator comparison. The two analyses with PTB units reported in Tab. 3 use the same observations, baseline predictors, model specification, and cross-validation folds, and differ only in how contextual surprisal is estimated. We enumerate all 4,096 sign patterns for the paired sign-flip tests over the 12 held-out trials. The two-sided p-values are 0.188 for first-fixation duration, 0.134 for gaze duration, and 0.620 for total reading time.

J DNA-TO-AMINO-ACID VALIDATION

We complement the scaling results in §4.2 by comparing estimates for the bigram and neural DNA models with exact target prefix probabilities obtained by brute-force enumeration of the precover. We use three targets of lengths 10, 13, and 15. The bigram model is trained on 2,000 synthetic coding sequences, each a start codon, a geometric number of codons (mean 30) sampled from human codon-usage frequencies, and a stop codon. We use the neural DNA model released with Snæbjarnarson et al. (2026), trained on simecek/Human_DNA_v0. beam_summing uses prune$_\tau$ with $\tau = 10^{-3}$. With the candidate cap of Snæbjarnarson et al. (2026) at 5,000 or 10,000, beam summing takes seconds instead of minutes on these targets but discards 87 to 99.9 percent of the mass under the bigram model and 52 to 95 percent under the neural model. The prior work's adaptive threshold settings change these estimates by less than 0.01 nats.

At M=256, beam_swor_ad has lower relative RMSE than smc_rb under both source models (Fig. 16). For the length-15 target under the neural model, the relative RMSE is 0.057 versus 0.156. Even at M=1024, smc_rb has higher relative RMSE (0.081) and takes 85.0 seconds per run, compared with 35.7 seconds for beam_swor_ad at M=256.

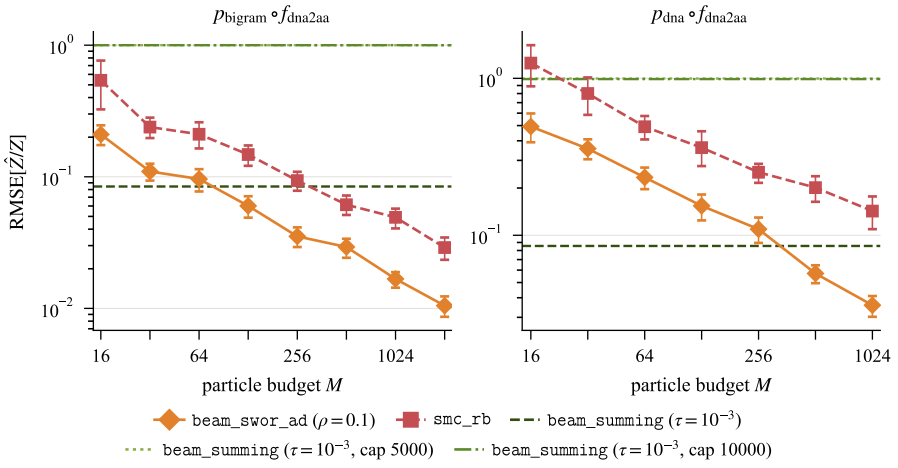


Figure 16: DNA-to-amino-acid validation. Corpus relative RMSE over 64 seeds, including zero estimates, with 95% bootstrap intervals. For each seed, we multiply the prefix-probability estimates for the three targets (lengths 10, 13, and 15) and compare with the product of their exact probabilities. The error of beam_swor_ad decreases faster with M than that of smc_rb, while beam_summing with prune$_\tau$ has fixed error. With a candidate cap of 5,000 or 10,000, beam_summing has relative error close to one in both panels.

K ADDITIONAL BYTE-TO-WORD RESULTS

Figs. 17 and 18 extend the exact bigram validation of Fig. 4 to every byte position in all ten paragraphs. Tab. 4 reports the additional per-paragraph GPT-2 estimates and runtimes.

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 pin each single-threaded run to one core on an otherwise idle machine and report its total CPU time. We verified that this was nearly identical to wall time. For GPT-2 we report wall time with one job at a time on the GPU. Following Snæbjarnarson et al. (2026), we convert GPT-2 Large to a byte-level model with the beam approximation of Vieira et al. (2025), using beam size 8 and pruning threshold 10^{-3}, and apply $\mathtt{f}_{\mathrm{ptb}}$ to its byte strings.

Variance and tails. For the live twist, a fixed-size SWOR draw preserves the pool's total weight (§3.2). Since $\sum_i \min(1, c\, w^{(i)}) = m$, every non-certain survivor has weight $1/c \le W/m$, where W is the pool's total weight before the draw. Roulette additionally rescales surviving weights by $1/\mu$ in `swor_adaptive` or by 2 in the tail-roulette decorator. These factors can compound across positions, and we give no bound on the variance of the final estimate. We report variation over repeated runs. For each method and particle count, Tab. 5 also lists the fraction of the sum of estimates contributed by the largest run and the span of the pooled log estimate under leave-one-seed-out deletion. For the with-replacement methods, the largest single-run estimate accounts for 70 to 100 percent of the sum of estimates in every reported setting. For `beam_swor` and `beam_swor_ad`, this share never exceeds 14 percent, and deleting any one seed moves the pooled log estimate by at most 0.2 nats.

For the bigram text model, Tab. 5 gives the relative RMSE with zero estimates included, the median and maximum of the per-seed corpus ratio, the log standard deviation over nonzero estimates, and the runtime per seed. At $M{=}200$ most `beam_swor_ad` runs return a small fraction of the mass and a few return many times it, whereas the `beam_swor` ratios at the same M are spread around one. From $M{=}2000$ the `beam_swor_ad` ratios concentrate around one.

To isolate the effect of sampling without replacement, we compare two versions of `beam_summing`, one using `swor` and the other using multinomial resampling with replacement. Both enumerate all live children of each particle. Compare this to `smc_rb`, which only samples one. Before pruning, both versions combine particles at identical source prefixes by summing their weights. All other settings are unchanged. We estimate the final prefix probabilities of all ten WikiText paragraphs under the bigram model, using $M{=}200$ and 32 matched seeds per method. Against the exact references, `beam_swor` has lower relative RMSE on every paragraph. None of its 320 runs returns zero, whereas 71 of the 320 with-replacement runs do. Every with-replacement seed returns zero for at least one paragraph, so all 32 corpus estimates are zero. Their relative RMSE is 1.000, compared with 0.780 for `beam_swor`. The mean total CPU time for the ten paragraphs is 308.8 seconds per seed for `beam_swor` and 86.5 seconds for the with-replacement variant. This comparison holds the maximum particle count fixed, not runtime.

The estimates in Fig. 19 are shown for `smc_rb`, the lowest-variance tested method with replacement, and `beam_swor_ad`, the lowest-variance tested method without replacement (Tab. 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 paragraph 0. On the corpus, `smc_rb` has a pooled log estimate 4.7 nats below that of `beam_swor_ad`. Resampling the 32 runs of each method gives a 95% bootstrap interval of $[-0.3, 7.9]$ nats for this corpus difference. Most of the difference comes from paragraph 1, where one `beam_swor_ad` run lies 7.5 nats above the other runs and raises the pooled log estimate by about 4 nats.

To check the reproducibility of `beam_swor_ad`'s estimate for paragraph 1, we run it with 32 additional independent seeds at $M{=}800$ and $\rho{=}0.1$. The pooled log estimate is -556.48 nats, compared with -556.41 nats for the original runs.

Fig. 20 compares the corpus estimates against runtime. The boxes span the interquartile range with whiskers at 1.5 IQR and seeds beyond them drawn as dots.

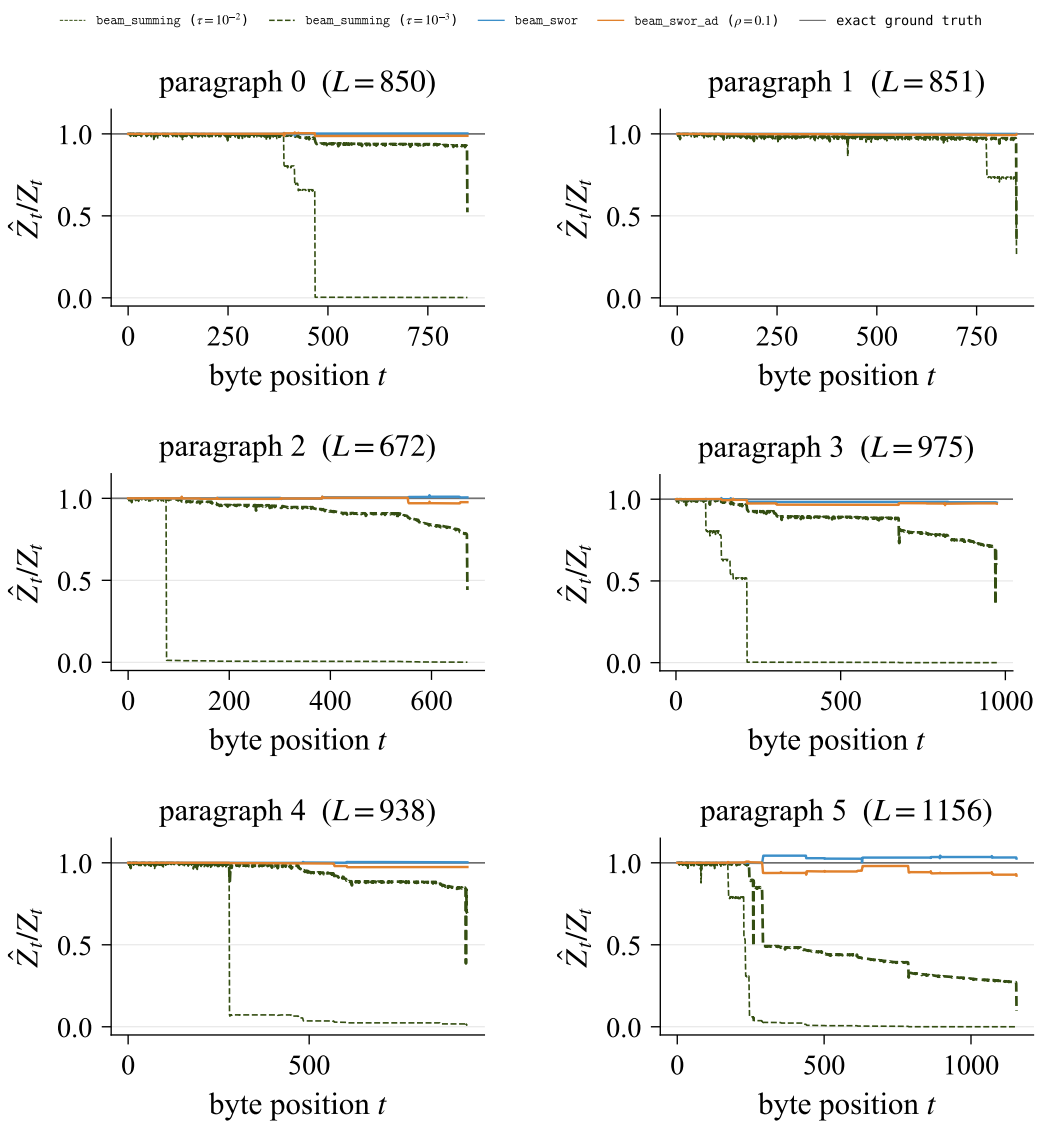


Figure 17: Per-position ratio $\hat{Z}_t/Z_t$ for all ten $p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$ paragraphs at $M{=}200$, averaged over 256 runs. Paragraphs 0–5 are shown here and 6–9 in Fig. 18. The `beam_swor` and `beam_swor_ad` averages remain close to one with sampling variation. Threshold-pruned beam summing at $\tau{=}10^{-2}$ and $\tau{=}10^{-3}$ falls below one where pruning discards positive covering mass.

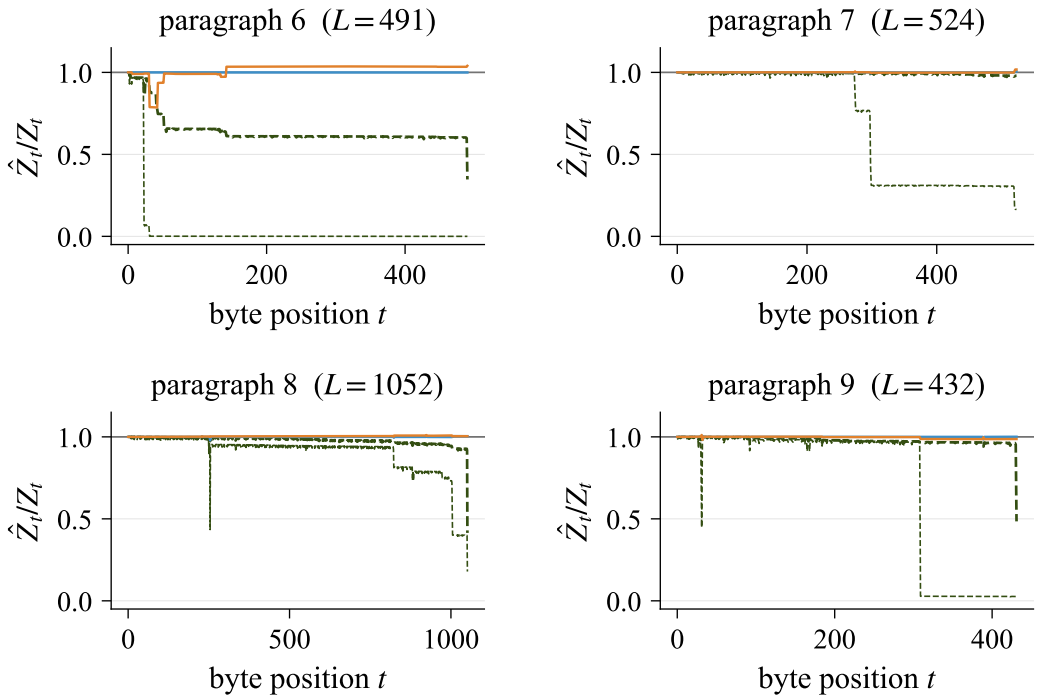


Figure 18: Per-position validation, paragraphs 6–9 (continued from Fig. 17), with the same methods and axes.

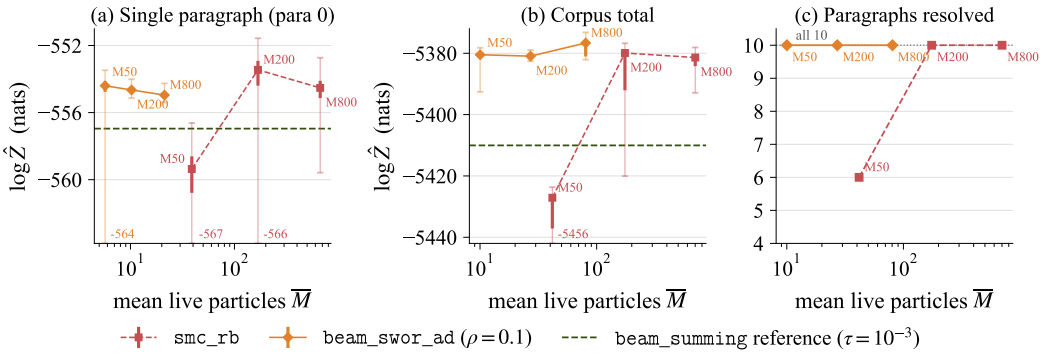


Figure 19: GPT-2 Large transduced with $\mathtt{f}_{\mathrm{ptb}}$. Panels (a) and (b) plot the estimates $\log \hat{Z}$ in nats against the mean live particle count. Labels give the maximum particle count M. Each point is $\log\big(32^{-1}\sum_{s=1}^{32}\hat{Z}_s\big)$, with a failed run contributing $\hat{Z}_s = 0$. 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 lower bound is 2.0 nats below the `beam_swor_ad` estimate on paragraph 0 and approximately 33 nats below it on the corpus. Since exact probabilities are unavailable, panel (c) counts the paragraphs on which each method recovers at least the lower bound's mass.

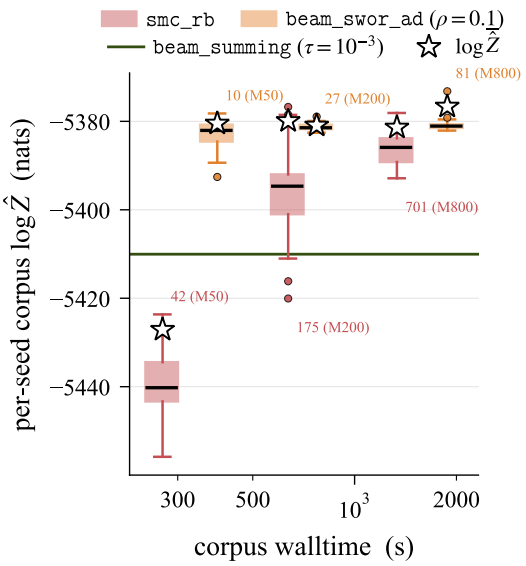


Figure 20: GPT-2 estimates versus runtime. Boxplots of the finite per-seed corpus log-estimates over 32 seeds. Black bars mark medians, and stars show $\log \bar{\hat{Z}}$, averaging $\hat{Z}$ over all seeds, including zeros, before taking the logarithm. The horizontal line is the corpus log-estimate from `beam_summing` with $\texttt{prune}_\tau$. Labels give the mean live particle count, with M in parentheses.

L Relation to Prior Work

Shah & Kroese (2018) develop sequential Monte Carlo methods for fixed-length vectors using sampling without replacement and Horvitz–Thompson reweighting. They build the vectors one entry at a time, enumerating all possible extensions before retaining a reweighted subset. The prescribed vector length fixes the number of extension iterations.

We extend their argument to the recursive beam-summing computation for transduced language models (Snæbjarnarson et al., 2026). Here, the target prefix stays fixed while the walk extends source prefixes one symbol at a time, potentially requiring arbitrarily many extensions. The transducer determines which extensions can still cover the target prefix and when to accumulate quotient and remainder contributions. The weighted quotient and remainder entries returned by the walk are then used to compute the next target-prefix probability.

The reweighting must therefore preserve expected contributions both within each source-prefix walk and across successive target positions. Theorem 3.1 establishes unbiasedness of this recursive estimator provided the run halts almost surely. Bounding the particle count alone does not ensure termination. Our adaptive pruning function uses the accumulated mass to control the particle count and guarantees almost-sure halting under the live twist (Proposition 3.2). The same tools address different problems in other settings, including SWOR for translation and search (Kool et al., 2019; Shi et al., 2020; Meister et al., 2021) and SMC for constrained language-model generation (Lew et al., 2023; Loula et al., 2025).

Table 4: Per-paragraph log-prefix probabilities in nats and runtime in seconds under $p_{\mathsf{gpt2}} \circ \mathsf{f}_{\mathrm{ptb}}$. `beam_summing` uses $\tau = 10^{-3}$. `beam_swor_ad` uses $\rho = 0.1$, nominal $M = 800$, and thirty-two runs. Parentheses give the between-run standard deviation, and runtimes are per run. The Δ column is `beam_summing` minus `beam_swor_ad`. The total row sums the per-paragraph log estimates. The corpus panel of Fig. 19 instead averages the per-run corpus estimates before taking the logarithm, hence its -5376.7.

para	beam_summing	beam_swor_ad	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

Table 5: Per-seed accuracy and cost on the ten-paragraph $p_{\mathsf{bigram}} \circ \mathsf{f}_{\mathrm{ptb}}$ corpus. Relative RMSE, median, and maximum describe the per-seed corpus ratio $\hat{Z}/Z$, including zero estimates. Top is the largest estimate divided by the sum of estimates. LOO is the span of the pooled log estimate under leave-one-seed-out deletion, in nats. The standard deviation of $\log \hat{Z}$ is computed over nonzero estimates, in nats, and runtime is per seed. Dead counts seeds with a zero estimate on some paragraph. Relative RMSE is close to 1 when every seed recovers essentially no mass and exceeds 1 only through overestimates. Settings at $M \leq 50$ have relative RMSE at least 1 and are omitted.

Algorithm	M	seeds	dead	rel. RMSE	median	max	top	LOO	std $\log \hat{Z}$	time (s)
smc_simple	200	32	0	1.00	0.00	0.00	1.00	9.81	44.5	353
smc_simple	800	32	0	1.00	0.00	0.00	0.99	5.18	17.3	1080
smc_simple	2000	32	0	1.00	0.00	0.00	0.70	1.19	10.4	2495
smc_rb	200	32	31	1.00	0.00	0.00	1.00	0.00	—	124
smc_rb	800	32	10	1.00	0.00	0.00	0.89	2.18	12.8	267
smc_rb	2000	32	2	1.00	0.00	0.03	0.88	2.16	7.03	585
smc_rb	4000	32	0	38.1	0.00	216.54	0.97	3.42	7.08	957
beam_swor	200	256	0	1.05	0.70	6.77	0.03	0.03	2.27	345
beam_swor	800	32	0	0.30	1.05	2.21	0.06	0.05	0.24	1390
beam_swor	2000	32	0	0.20	0.98	1.51	0.05	0.02	0.20	3432
beam_swor_ad	200	256	1	2.89	0.00	20.65	0.12	0.13	6.47	52
beam_swor_ad	800	32	0	0.71	0.68	3.37	0.14	0.15	2.14	115
beam_swor_ad	2000	64	0	0.49	0.94	2.82	0.04	0.04	0.45	196
beam_swor_ad	4000	32	0	0.31	0.96	1.72	0.05	0.04	0.32	346
beam_swor_ad	8000	256	0	0.16	0.97	1.57	0.01	0.00	0.16	623