arXiv:2611.32855v1 [cs.IR] 24 Nov 2026

Speculative Decoding for Semantic-ID-Based Generative Recommendations

Harshal A. Chaudhari [1]

Abstract

Generative recommender systems with Semantic IDs (SIDs) represent each item as a short sequence of L discrete tokens and use a sequence model to predict the next item's tokens from a user's history. Industry-scale deployments typically use these models to retrieve multiple candidates per request with constrained beam search. However, this incurs a fixed cost of L sequential model calls per request, and accuracy degrades as the SID generator becomes less sharp, without a guarantee on how close the output is to the exact top-K. Production systems therefore cannot detect degraded retrieval for individual requests.

We propose SIDECAR, which uses approximate draft candidates and a tree-attention pass to score the entire draft in a single model call, memory permitting. This pass bounds the probability of every unexplored SID prefix and certifies the exact top-K if the draft contains them. Otherwise, a branch-and-bound fallback expands only the parts of the trie that can still change the output, reaching certification within at most L calls for any draft. Moreover, under a budget of M tokens per call and Y calls, which for $Y = L$ matches the resources of a beam search of width M, SIDECAR announces a factor $1 + \delta$ within which its list will be exact at every rank. Depending on model sharpness, SIDECAR returns the exact top-K using a fraction of beam search's calls or memory. Even when the budget is too small to recover the exact top-K, SIDECAR, unlike beam search, reports the announced factor, allowing the serving system to stop early once a quality target or latency limit is reached.

	Exhaustive scoring	Beam-search decoders	Speculative decoders	SIDECAR (ours)
Exact top-K	✓	∼	∼	✓
Exactness certificate	✓	✗	✗	✓
Early stop	N/A	✗	✗	✓
Guarantee under a budget	N/A	✗	✗	✓
Sequential model calls	1	L	$< L$	∼1–2 (worst case L)
Memory per call	catalog	W	W + draft	M ($\leq W$)
Draft	—	—	trained or retrieved draft	any draft candidates
Examples	—	TIGER, GR4AD, UtilityGR	AtSpeed, NEZHA, SpecGR	—

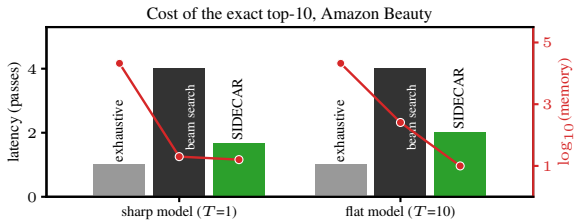


Figure 1. SIDECAR returns the top-K with an exactness certificate, in fewer passes and less memory per pass than beam-search decoders.

1. Introduction

Semantic IDs (SIDs) have become a common way for large-scale recommender systems to represent their items: a residual quantizer maps the content of each item to L discrete codes (Lee et al., 2022; Rajput et al., 2023), and a sequence model emits these codes one after another, as generative retrieval does for documents (Tay et al., 2022; Wang et al., 2022). Thus, the model does not need an output layer of the size of the catalog, and a new item receives an identifier from its content alone. Following TIGER, most research has focused on generating these codes, learning them with collaborative signals (Jin et al., 2024; Wang et al., 2024a; Liu et al., 2025), and several companies now serve this design in production (He et al., 2026; D'Amico et al., 2026; Deng et al., 2025; Zhai et al., 2024).

Decoding has also received attention; however, most of this work changes the model to suit the decoder, with multi-token heads (Hou et al., 2025; Zhan et al., 2026), trained drafters or self-drafting heads (Lin et al., 2025; Wang et al., 2026b), or scorers retrained for beam search (Zhuo et al., 2020). A model directly served after training is still decoded with constrained beam search over the trie of valid SIDs (Ju et al., 2025), and its large-scale variants adapt the width (Xue et al., 2026), split the beam (Wang et al., 2026a), or rescore prefixes (Hu et al., 2026), still running one dependent pass per SID level. This comes at a measurable cost: a single head decoded with beam search produces better rankings and content quality, whereas multi-token prediction provides higher throughput (Oh et al., 2026). We therefore ask the

[1]Etsy Inc., Brooklyn, NY, USA. Correspondence to: Harshal A. Chaudhari <hchaudhari@etsy.com>.

Preprint. November 25, 2026.

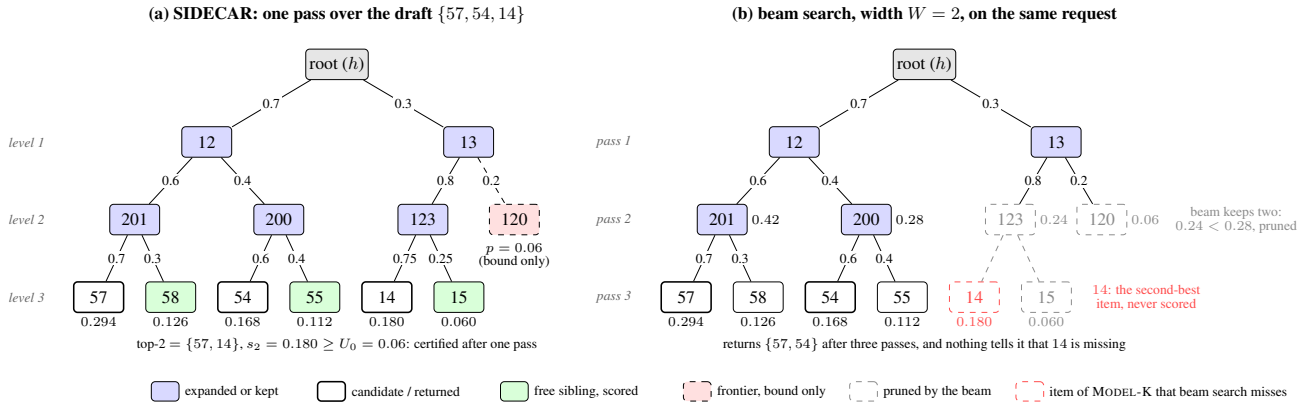


Figure 2. The same request under the two decoders ($L = 3$, $K = 2$). **(a)** One pass over the draft scores the candidates and their free siblings, and the only unexpanded prefix weighs less than the second-best score, so the first pass is certified. **(b)** Beam search of width K keeps the two heaviest prefixes per level and prunes $(13, 123)$, which holds the second-best item. It returns a wrong list after three passes, and nothing in its computation reveals the error.

following question: can we design a decoder that returns the exact output of a single head while avoiding its sequential cost?

Beam search spends memory and latency, the two constrained resources of large-scale production systems, in a rigid way: on every request it runs L passes of W prefixes each and outputs a list only after the last pass. Yet most of these passes do not change the final output. The SID codebooks address far more cells than most catalog sizes, so below the first levels most prefixes contain single items. Beam search nevertheless spends a full sequential pass on each of the remaining codes of such prefixes. This approach also constrains the design of the codes themselves. Since every level adds a dependent pass, SIDs are usually kept short (Hou et al., 2025; Xie et al., 2026), and recent industrial work moves capacity from depth to a larger codebook (Xie et al., 2026). A larger codebook, however, makes the code space dwarf the catalog even more. Whether deeper trees would capture the embedding space better remains open. Moreover, beam search cannot tell which of its output lists differ from the exact top-K of the model. Its output list is exact when W exceeds the number of prefixes above the K-th best item at every level, which beam search cannot verify at runtime. In Fig. 2(b), for example, a beam of width K prunes the prefix that holds the second-best item, and nothing in the computation of beam search reveals that its list is therefore wrong. Both limitations above, namely the fixed cost of L passes and the missing certificate, stem from the decoder rather than from the model itself. Hence, we keep the model and its codes unchanged and redesign only the decoder.

By design, beam search assumes that nothing is known about the final output before its first pass. However, most large-scale recommender systems in the industry can easily provide a draft of the answer. Most contemporary production stacks run a first-stage retriever before the ranker, and candidates are often computed offline using batched jobs. Alternatively, a batch job can store the top-N items of the model itself, or of a cheaper, heavily quantized copy of the model that is exported at training time. The batch job can compute a high-quality draft for each user from an interaction history that is a few interactions old. At request time, the decoder can then retrieve such drafts through KV stores. Furthermore, one pass with a tree-attention mask (Miao et al., 2024; Cai et al., 2024) can score the whole draft, just as speculative decoders verify a draft tree (Leviathan et al., 2023). In this paper, we introduce SIDECAR (Semantic-ID Decoding with Exactness Certificates from Approximate Retrieval drafts), a decoder that exploits such drafts. Because the decoding is constrained to the trie of valid SIDs (De Cao et al., 2021), the pass that scores the draft also bounds every item outside it. Hence, SIDECAR can prove that its list is the exact top-K (Fig. 2(a)), which no existing speculative decoder for SIDs does (Lin et al., 2025; Ding et al., 2026), and when the certification fails, a fallback expands only the prefixes that can further improve the output list. Thus, given enough memory per pass, the output is exact for every draft. The draft quality itself decides only how many passes are required. Moreover, SIDECAR uses the structure of the catalog trie, which beam search ignores, to announce how close to exact its output will be as early as after the first pass. Finally, since SIDECAR does not pay one pass per SID level on every request, we argue that it opens new avenues for deeper SID trees, which can capture richer semantic structures from the item embeddings.

We develop the theory of SIDECAR in two stages. In Sections 2 and 3, we assume unbounded memory per pass, where the certificate, the fallback, and the role of the catalog trie are simplest to state. Building upon this intuition, Section 4 adds the memory and latency budgets of serving, under which BUDGET-SIDECAR plans its passes so that its list comes with a worst-case guarantee within the budget. Our experiments (Section 5) compare the two decoders at

beam search's width and its L passes, across models of increasing flatness. On a sharp model, the decoders return the same lists, but SIDECAR needs less than half of the passes at equal memory and certifies every exact list, whereas beam search certifies none. As the model flattens, beam search's correctness collapses at every width short of the whole catalog, because it ranks prefixes by their probability mass, whereas the top items of a flat model lie in clusters with little mass. SIDECAR, in contrast, keeps returning the exact list with a fraction of beam search's memory, since its draft points directly to the top items, whereas beam search has to keep every heavy cluster to reach them. From half of beam search's width on, SIDECAR also certifies that its list is exact. Finally, the announcement after the first call tells the deployment which of these cases it is in, and it states the guarantee factor where neither resource suffices, whereas beam search gives no indication of how far its list is from the exact one. Figure 1 compares SIDECAR with exhaustive scoring, beam-search decoders, and speculative decoders. Appendix A provides the assumptions and the proofs of all formal results.

2. The decoder and its certificate

In this section, we describe the setting of constrained decoding and provide the necessary notation. Then, we present SIDECAR with unbounded memory per pass, namely its first pass, its certificate, and its fallback, and we show how the quality of the draft affects the cost.

Constrained decoding over the catalog trie. Throughout the paper, every item e of the catalog $\mathcal{E}$ has a Semantic ID $\mathrm{SID}(e) = (c_1(e), \ldots, c_L(e))$, where the first $L-1$ codes come from the content of the item and the last code separates items with equal content codes. The SIDs of the catalog form the *catalog trie* $\mathcal{T}$, which the serving stack holds. Given the user history h and a prefix $c_{<\ell}$, the logits z of the model define, as in constrained decoding, probabilities over the admissible next codes only:

$$p(c_\ell \mid h, c_{<\ell}) = \frac{\exp z_{c_\ell}}{\sum_{c \in \mathrm{children}(c_{<\ell})} \exp z_c}, \qquad (1)$$

where $\mathrm{children}(c_{<\ell})$ denotes the admissible children of the prefix in $\mathcal{T}$. The probability of an item is the product of its L conditionals, and the probability of a prefix b is the product of the first $\mathrm{depth}(b)$ of them. Since the conditionals at a prefix sum to one over its children, the probability of a prefix equals the total probability of the items below it.

Our target is the exact top-K of the model, i.e., the K items of highest probability under the model for the request at hand, which we denote by MODEL-K. Every decoder returns a list of K items; constrained beam search returns BEAM-K, the top-K of its final beam after L sequential passes with W prefixes per level, which only approximates MODEL-K. Our decoder returns SIDECAR-K. We use SIDECAR-K$_r$ to denote its output list after round r of the procedure below. We call a returned list *exact* when it equals MODEL-K. SIDECAR-K is *within a factor* $1+\delta$ when every item of MODEL-K is within that factor of the item at its rank.

The first pass. A *draft* is any set C of N candidate items. We map the candidates to their SIDs and insert them into a trie, henceforth referred to as the *candidate trie*. The internal nodes follow the history in the input, and a tree-attention mask lets every node attend to the history, to its ancestors, and to itself. With position ids that follow the depth of a node, the output at every node equals the output that a separate pass over its own path would produce. Hence, one pass returns the model's distribution at every internal node of the candidate trie. From these outputs, we read two kinds of values. First, every item whose depth-$(L-1)$ prefix is in the trie receives its exact probability, which includes the candidates and also their siblings at no extra cost. Second, every admissible child of an expanded node that is not itself expanded is a *frontier* prefix, and the pass returns its probability as well, which bounds every item below it. Figure 2 illustrates the pass on a small example.

After any round r, let E_r denote the set of items with exact probabilities, $s_K(r)$ the K-th largest of these probabilities, F_r the frontier, and $U_r = \max_{b \in F_r} p(b \mid h)$ the largest frontier probability. The list SIDECAR-K$_r$ is the top-K of E_r. Since every item outside E_r lies below exactly one frontier prefix, its probability is at most U_r, which gives the following certificate.

Theorem 1 (certificate)**.** *If* $s_K(r) \geq U_r$, *then* SIDECAR-K$_r$ = MODEL-K.

When the condition of Theorem 1 holds, we call SIDECAR-K *certified*. Figures and tables name the two outcomes, exact and certified, EXACT and CERTIFICATE respectively. Note that BEAM-K can be exact on some requests but never certified, because beam search cannot discern its exact lists from its other lists.

The test uses only quantities that the completed passes have already computed, namely a sort of the exact scores and a maximum over the frontier. In other words, the draft decides only how soon the certification test passes. If it contains the exact top-K, the test passes after the first pass. Otherwise, the fallback rounds continue until the test passes, and the returned list is still the exact top-K.

The fallback. If the certification fails after the first pass, some frontier prefix may hide a better item underneath. We call a frontier prefix *open* when its probability exceeds the current K-th best score, and we denote the open set by $O_r = \{b \in F_r : p(b \mid h) > s_K(r)\}$. The next round only expands the open prefixes, in one pass that reuses the cached

Algorithm 1 SIDECAR

Require: history h, draft C, list size K
$E_0, F_0 \leftarrow$ one tree-attention pass over the candidate trie of C after h
for $r = 0, 1, \ldots, L-2$ **do**
  $O_r \leftarrow \{b \in F_r : p(b \mid h) > s_K(r)\}$
  if $O_r = \emptyset$ **then**
    return top-K of E_r
  end if
  $E_{r+1}, F_{r+1} \leftarrow$ one pass over the children of O_r with the cached KV
end for
return top-K of E_{L-1}

history and ancestors, and reads their children. Algorithm 1 summarizes the procedure.

Proposition 1 (termination)**.** *Regardless of the draft and of the model, s_K never decreases and U never increases from one round to the next, and Algorithm 1 returns* MODEL-K *after at most L passes.*

Intuitively, each round descends one level of the trie below the open prefixes, so after $L-1$ rounds nothing can remain open. Thus, a good draft lets the procedure stop after the first pass. A poor draft requires more rounds, but the list is still exact.

Early stopping before the certificate holds. A deployment may stop the procedure before the certificate holds, for instance at a cap on the number of passes that fit under the maximum serving latency. The list at that point then carries a weaker but valid statement. Let $s_k(r)$ denote the k-th highest exact score and $p^*_{(k)}$ the probability of the model's k-th best item, and define the *gap* as $\mathrm{gap}_r = U_r / s_K(r) - 1$.

Proposition 2 (per-rank bound)**.** *After any round r and for every rank $k \le K$,*

$$s_k(r) \le p^*_{(k)} \le \max\big(s_k(r), U_r\big) \le \big(1 + \max(\mathrm{gap}_r, 0)\big)\, s_k(r).$$

Hence, SIDECAR-K$_r$ is within a factor $1 + \mathrm{gap}_r$ of MODEL-K at every rank, and $\mathrm{gap}_r \le 0$ is exactly the certificate. Note that beam search offers neither statement, since it keeps no bound on the items it prunes.

Effect of draft quality. The quality of the draft never affects exactness. Instead, it affects the number of passes, and it does so through one quantity, namely the probability that the first pass leaves unscored, $\varepsilon = 1 - p(E_0)$, which we call the *hidden probability*. If $\varepsilon < p^*_{(K)}$, i.e., if the hidden probability is below the probability of the model's K-th best item, then SIDECAR-K$_0$ is certified. To see why, note that the frontier prefixes of the first pass are disjoint and hold exactly the unscored probability, so each of them has probability at most ε. An unscored item of MODEL-K would lie below one of them with probability at least $p^*_{(K)} > \varepsilon$, which is impossible. Hence, every item of MODEL-K is scored, $s_K(0) = p^*_{(K)} > \varepsilon \ge U_0$, and the certificate holds. Thus, a draft does not have to contain MODEL-K. It only has to contain items from the right prefixes, so that the probability it leaves uncovered is smaller than that of the K-th best item.

3. Exploiting the structure of the catalog trie

Building upon the decoder of Section 2, this section shows what the structure of the catalog trie contributes to SIDECAR. First, we observe that most deep prefixes of a SID trie hold a single item. Then, we use this structure to save tokens in every pass and to resolve some open prefixes in a single round.

Single-item prefixes. In a SID trie, most deep prefixes hold a single item, and this follows from the size of the code space rather than from any one catalog or quantizer. The $L-1$ content codes, drawn from codebooks of size B, address B^{L-1} cells, which is orders of magnitude more than the thousands to millions of items that a catalog holds (e.g., 16.7 million cells with the three codebooks of 256 codes of TIGER (Rajput et al., 2023)). Moreover, every quantizer in use, whether a residual quantizer (Lee et al., 2022; Deng et al., 2025) or a hierarchical clustering of the item embeddings (Tay et al., 2022; Wang et al., 2022), spreads the items of a prefix over its codebook at every level, so that the number of items per prefix falls geometrically with depth. Indeed, the collision code of TIGER exists because the content codes alone leave almost every item in a prefix of its own (Rajput et al., 2023). Thus, below the first level, most prefixes hold a single item. At such a prefix, the model has one admissible child, so its constrained conditional is one and the item below inherits the probability of the prefix. Consequently, these prefixes need no pass and no token.

We call a prefix *branching* when it has at least two admissible children. The *skeleton* of a prefix b, denoted by $S(b)$, is the set of branching prefixes below b together with the non-branching prefixes on the paths that connect them to b, which we call *connectors*. Note that the connectors are needed because a node can only be fed together with its ancestors. We use $\sigma(b) = |S(b)|$ to denote the number of nodes in the skeleton, i.e., the number of tokens that a pass has to feed to resolve every item below b at once. Both $S(b)$ and $\sigma(b)$ depend only on the catalog trie, so they are known before any request arrives, and the outputs at the skeleton determine the probability of every item below b. Since most deep prefixes are chains, the skeleton of a cluster is a small fraction of its prefixes, and the skeleton of the whole catalog is a small fraction of the trie.

Chain rules. We exploit the chains in three places of SIDECAR, and in each of them we feed only nodes that are known to be needed. First, in the first pass, a candidate prefix whose subtree is a chain contributes nothing to the scores, because the conditional at its parent gives its probability, and every node on the chain passes this probability unchanged to the item. Hence, the first pass feeds only the branching candidate prefixes and the connectors on the candidate paths. Second, a frontier prefix whose subtree is a chain is resolved without a token for the same reason, since its item receives the probability of the prefix, which the pass has already returned. Third, an open prefix with a single admissible child makes that child open too, since the child inherits its probability. Thus, a fallback round feeds the whole unary run down to the next branching prefix at once, instead of one node per round. These rules leave the certificate, the exactness, and the bound of L passes untouched.

The skeleton round. A fallback round of Algorithm 1 feeds the open prefixes and reads the conditionals of their children, which become the frontier of the next round. Hence, the fallback descends one level per round, and this is where the bound of $L-1$ rounds comes from. When the memory per pass is unbounded, the tree pass can instead feed an open prefix together with its whole skeleton, exactly as the first pass feeds the whole candidate trie. The ancestors of every fed node are then in the sequence, as tree attention requires, and consequently every item below the open prefix has an exact probability after the pass. Hence, if the first fallback round feeds every open prefix $b \in O_0$ together with its skeleton, at a cost of $\sum_{b \in O_0} \sigma(b)$ tokens, nothing below O_0 remains on the frontier, and every other frontier prefix stays closed, since s_K does not decrease. Thus, the open set is empty after the second pass, and SIDECAR-K is certified, regardless of the draft and of the model.

In other words, the skeleton round saves rounds at the cost of additional tokens. Under a bound on the nodes of a pass, however, the skeletons of all open prefixes may not fit into one pass. In the memory-bound regime of serving, a pass of a few thousand tokens costs little more than a pass of fifty, so the trade favors the skeleton whenever the open set is a moderate number of clusters. Even in the worst case over all drafts, with every cluster open, the round feeds the skeleton of the catalog, which is a fraction of the nodes that an exhaustive pass over the trie would feed. Unlike the chain rules, the skeleton round feeds nodes before knowing that they are open, so it is not part of the default decoder.

4. Decoding under a budget

So far, we have assumed that a pass can hold as many nodes as the draft and the fallback need. In practice, however, a serving deployment provisions its resources in advance and applies the same limits to every request. It bounds the memory of each pass by the number of nodes that a pass may hold, denoted by M, and the latency of each request by the maximum number of passes that it may run, denoted by Y. Note that for $Y = L$ these are exactly the resources of constrained beam search with width $W = M$. In this section, we address two questions about such a budget. First, what can SIDECAR guarantee when the budget may be too small for exactness? Second, how soon can we provide this guarantee, i.e., before spending the remaining budget? The guarantee takes the form of an approximation factor: SIDECAR-K is within a factor $1+\delta$ of MODEL-K at every rank, in the sense of Proposition 2. We call the smallest δ that can be stated after the first pass the *announced factor*. We refer to SIDECAR under such a budget as BUDGET-SIDECAR (Algorithm 2), and to its execution as the *planned run*. First, we describe what the first pass leaves us with (line 1 of Algorithm 2). Second, we bound what resolving one open prefix can cost. Third, we show how the costs of all open prefixes translate into a number of rounds and into a plan for each round (lines 5–7). Finally, we derive the factor that the rounds within the budget can guarantee in the worst case (line 2).

Algorithm 2 BUDGET-SIDECAR: SIDECAR with M nodes per pass, Y passes, and a target factor $1+\delta_t$ set by the deployment

Require: history h, draft C, size K, target factor $1+\delta_t$ (default $\delta_t = 0$, i.e., exact)
1: $E_0, F_0 \leftarrow$ first pass over the candidate trie of C: at most M skeleton members, in draft order
2: **announce** the factor $1 + \max(\delta^*(M, Y), \delta_t)$ for Y passes, and $Y_{\min}$ passes for exactness
3: **for** $r = 0, 1, \ldots, Y-2$ **do**
4: **if** no prefix of F_r is open **then return** top-K of E_r
5: re-plan from F_r: factor in force Δ_r, pool, plans and schedule x_r
6: $B \leftarrow$ the heaviest open prefixes, at most M
7: add pool nodes until B holds x_r of them, displacing the lightest prefixes outside the pool, then fill the free slots with more
8: $E_{r+1}, F_{r+1} \leftarrow$ one pass over B with the cached KV
9: **end for**
10: **return** top-K of E_{Y-1}

Throughout this section, we follow our established conventions. First, the state after round r carries the index r, i.e., E_r is the set of scored items, $s_K(r)$ the K-th best exact score, and F_r the frontier. Round 0 is the first pass. Hence, after round r, the planned run has used $1+r$ passes, and $Y-1-r$ fallback rounds remain. We state the construction below for $r = 0$, which is the moment of the announcement, and line 5 of Algorithm 2 repeats it after every round. Second, the guarantee is a worst case over what the first pass has not observed. After the first pass, the decoder knows the outputs of the model at the fed nodes, but not how the proba-

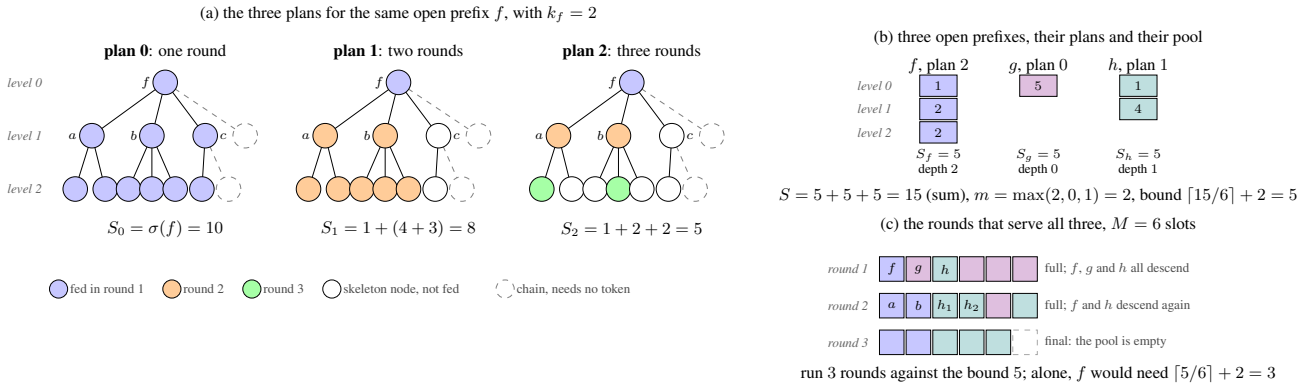


Figure 3. Plans and rounds under a budget (Propositions 3 and 4). **(a)** The three plans of one open prefix with $k_f = 2$, so that at most two nodes per level can be open: plan 0 feeds the whole skeleton in one round, plan 2 one level per round, plan 1 lies between, and the adversary opens the nodes with the largest skeletons. **(b)** Three open prefixes with different plans: the pool size S adds over the prefixes, whereas the depth m is their maximum, since they descend together. **(c)** Six slots per round serve all three in three rounds, known-open prefixes first and skeleton nodes after, which beats the bound of five because no round was under-filled while depth remained.

bility of each frontier prefix splits among the prefixes below it. We call any assignment of these unobserved outputs that agrees with the first pass a *consistent completion*, and we let an *adversary* choose the completion that hurts the fallback most. The announced factor holds against this adversary, and consequently for every consistent completion. Since the real model is one of the consistent completions, the factor holds for it as well, usually with much room to spare.

Threshold and open prefixes. The first pass (line 1 of Algorithm 2) is the first pass of SIDECAR capped at M nodes, i.e., it feeds the skeleton members of the candidate trie in draft order for as long as they fit. It returns the scored items E_0, their K-th best score $s_K(0)$, and the frontier F_0, together with the probability $p(f)$ of every frontier prefix f. Below the frontier, however, nothing is yet known. From these quantities, we can determine whether the current list is within a given factor of MODEL-K. Every unscored item lies below some frontier prefix, so its probability is at most that of the prefix. Hence, SIDECAR-K is within the factor $1 + \delta$ as soon as every frontier prefix b satisfies $p(b) \le (1 + \delta)\, s_K$, where s_K is the current K-th best exact score. Let $t = (1 + \delta)\, s_K(0)$ denote this *threshold* after the first pass. We call a frontier prefix *open* at t when $p(b) > t$. Since s_K never decreases, a prefix that is closed at t stays closed. Thus, the task of the fallback is to bring every open prefix below t within the allowed rounds and nodes. Note that the announcement is made before the fallback runs, so it can use only the quantities returned by the first pass and the catalog trie.

The skeleton of a frontier prefix b also tells how many levels of expansion resolve it under the one-level-per-round fallback, namely one plus the depth of its deepest branching prefix minus the depth of b. We call this number the *resolution depth* $d_b \le L - \mathrm{depth}(b)$.

Plans for one open prefix. Fix an open prefix $f \in F_0$. The fallback has to find out whether some prefix below f is also above t, and it has to keep doing so until no such prefix remains. We count *levels* from f: f itself is at level 0, its children are at level 1, and its descendants j levels down are at level j. Two facts bound the number of prefixes at level j that can be above t, and both of them are known after the first pass. First, the prefixes at any one level below f are disjoint and have total probability $p(f)$, so at most $k_f = \lfloor p(f)/t \rfloor$ of them can exceed t. Second, only skeleton nodes need to be fed (Section 3), and only $c_j(f)$ of them exist at level j, where $c_0(f) = 1$ counts f itself and $\sigma(f) = \sum_j c_j(f)$ is the size of the skeleton of f. Within these two bounds, however, nothing observed so far tells which nodes are open, so we let the adversary decide.

The fallback can resolve f in two ways. The first way is *level by level*: the fallback feeds f, reads its children, feeds the open ones among them, and so on, descending one level per round. Every node fed in this way is known to be open. However, each level requires a separate round, so resolving f takes up to d_f rounds, where d_f is its resolution depth. The second way is the *whole skeleton*: the fallback feeds f together with its entire skeleton in one round. This takes a single round, but it costs $\sigma(f)$ nodes whether or not they turn out to be open. Note that feeding only part of a level does not help against the adversary, who can place the hidden probability on whatever part of the level was skipped. Therefore, the options that admit a worst-case guarantee are the mixtures of the two ways, in which the fallback descends ℓ levels one round at a time and then feeds the open nodes of level ℓ together with their whole skeletons. We call $\ell_f \in \{0, \dots, L - 1 - \mathrm{depth}(f)\}$ the *plan* of f, and Fig. 3(a) shows the three plans of one open prefix.

Proposition 3 (worst-case cost of a plan)**.** *Let f be open at threshold t and $k_f = \lfloor p(f)/t \rfloor$. For every consistent completion:*

(i) *at most* $\min(c_j(f), k_f)$ *nodes at level* j *below* f *are above* t *and have to be fed;*

(ii) *the plan* ℓ *feeds at most*

$$S_\ell(f,t) \;=\; \underbrace{\sum_{j=0}^{\ell-1} \min\big(c_j(f), k_f\big)}_{\text{levels fed one at a time}} \;+\; \underbrace{\Sigma_\ell^\sigma(f,t)}_{\text{skeletons at level } \ell} \qquad (2)$$

nodes before nothing below f *is open, where* $\Sigma_\ell^\sigma(f,t)$ *is the sum of the* k_f *largest skeleton sizes among the nodes at level* ℓ*.*

The plan $\ell = 0$ is the skeleton round, at a cost of $\sigma(f)$ nodes. The largest plan is pure level-by-level descent. It feeds only nodes that are known to be open, but it needs one round per level. Between these two extremes, a larger plan uses more rounds and feeds fewer nodes.

Combining the plans of all open prefixes. Suppose that every open prefix has been given a plan, and let the *pool* denote the set of nodes that these plans may feed. The pool has a worst-case size S, namely the sum of Eq. (2) over the open prefixes, and a depth $m = \max_f \ell_f$, i.e., the largest number of levels that any prefix descends one round at a time. Note that both S and m are fixed by the plans and the catalog before any fallback round runs. The planned run will discover which nodes are open. However, since a closed prefix stays closed, every prefix that becomes open later is a descendant of some $f \in F_0$, and the plan of f already provides for it.

Proposition 4 (round bound and schedule)**.** *Let the open prefixes of* F_0 *have plans with pool size* S *and depth* m*.*

(i) *The fallback finishes within*

$$\lceil S/M \rceil + m \qquad (3)$$

rounds, whatever the order in which full and under-filled rounds occur.

(ii) *If* $\lceil S/M \rceil + m \le y$ *with* y *rounds remaining, then*

$$x_r \;=\; S - (y - 1 - m)\, M \;\le\; M, \qquad (4)$$

and a round that feeds at least x_r *pool nodes, or every pool node that is available, leaves a pool with* $\lceil S'/M \rceil + m' \le y - 1$*.*

This bound counts two kinds of rounds. A round that fills all its M slots with pool nodes is *full*, and since the plans feed at most S pool nodes in total, there are at most $\lceil S/M \rceil$ full rounds. A round that cannot be filled has fed every pool node whose parent is already known. Hence, the depth that remains to be descended drops by one, and this can happen at most m times. Note that the two terms aggregate over the open prefixes in different ways. Nodes are not shared between open prefixes, so their counts add up to S, whereas rounds are shared, since all open prefixes descend in the same rounds. Hence, the depth term is the maximum m of the plans rather than their sum. Figure 3(b) and (c) show three open prefixes with different plans and the rounds that serve them together.

Part (ii) of Proposition 4 gives the rule for a single round, and lines 6 and 7 of Algorithm 2 carry it out. We call the number x_r the *schedule*, i.e., the number of pool nodes that the current round must feed so that the round bound still fits into the rounds that remain after this one. The pool nodes that are available in a round are of two kinds. The first kind are the open prefixes themselves. Every exact decoder has to feed them, so line 6 takes the heaviest of them that fit into the M slots. The second kind are the skeleton nodes below the prefixes with plan zero. At this point, nothing tells whether they are needed, so feeding them prepares for the worst case without knowing whether it occurs. Line 7 feeds such nodes only as far as the schedule requires: it adds them until the batch holds x_r pool nodes, and when the batch is already full, they replace the lightest of its prefixes that lie outside the pool. We call these nodes the *speculative nodes*, and they are the only nodes that the planned run ever feeds without knowing that they are needed. Finally, any slots that remain free are filled with further speculative nodes, since a pass takes about the same time whether or not its slots are filled, whereas a filled slot can resolve part of an open cluster.

The announced factor. The best plans are those that minimize the bound of Eq. (3), and this minimization decomposes over the open prefixes. To see why, fix a cap m on the depth. Then the depth term of the bound is at most m for any plans with $\ell \le m$, so every open prefix can independently take its cheapest plan with $\ell \le m$. For example, a heavy prefix with a small skeleton is fed whole, whereas a light prefix with a large skeleton descends level by level, and both happen in the same batch. Let $S_m^*(t) = \sum_{f \in F_0 :\, p(f) > t} \min_{\ell \le m} S_\ell(f,t)$ denote the resulting pool size. Minimizing over the cap then gives the smallest bound,

$$R^*(t) \;=\; \min_{0 \le m \le L-2} \big(\lceil S_m^*(t)/M \rceil + m \big). \qquad (5)$$

Note that everything in $R^*(t)$ is known after the first pass: the frontier probabilities give k_f, the catalog gives $c_j(f)$ and the skeleton sizes, and M is part of the budget. Moreover, a larger threshold opens fewer prefixes and gives them smaller counts k_f, so R^* is non-increasing in t. Hence, the announced factor is the smallest δ for which these rounds fit into the $Y - 1$ fallback rounds of the budget,

$$\delta^*(M,Y) \;=\; \min\big\{ \delta \ge 0 : R^*\big((1+\delta)\, s_K(0)\big) \le Y - 1 \big\}, \qquad (6)$$

and Appendix C explains how to compute it, either by bisection or exactly. Similarly, we write $Y_{\min}(M) = 1 + R^*(s_K(0))$ for the smallest Y with $\delta^*(M, Y) = 0$, i.e., the number of passes after which SIDECAR-K is certified with M nodes per pass. Both $\delta^*(M, Y)$ and $Y_{\min}(M)$ are known after the first pass, and line 2 of Algorithm 2 states them.

The planned run. A deployment may also hold a *target* δ_t, i.e., the factor that is good enough for it, with $\delta_t = 0$ for exactness. Two thresholds then govern each round. The first is the target threshold $(1 + \delta_t)\, s_K(r)$: a frontier prefix is *open* when it lies above this threshold, because it can then still change SIDECAR-K beyond what the target allows. Hence, line 4 of Algorithm 2 stops only when no prefix is open, and line 6 feeds the open prefixes first. The second is the threshold of the *factor in force* $\Delta_r = \max(\delta_r^*, \delta_t)$, where δ_r^* denotes Eq. (6) recomputed from F_r and $s_K(r)$ with $Y - 1 - r$ rounds remaining, so that $\delta_0^* = \delta^*(M, Y)$ is the announcement.

In each round, line 5 computes the pool, the plans, and the schedule only for the prefixes above $(1 + \Delta_r)\, s_K(r)$, since only these prefixes can violate the factor in force. When $\Delta_r > \delta_t$, the prefixes between the two thresholds are open but outside the pool: line 6 feeds them when they are among the heaviest, and these are the prefixes that line 7 may displace. When several sets of plans attain the same factor, the planned run uses the one with the smallest $S + mM$, which by Eq. (4) forces the least speculation. Finally, line 8 is a fallback pass of Algorithm 1, with the batch in place of the open set.

Comparison with the unplanned run. In order to see what the planning changes, consider a run without it. Without planning, SIDECAR under a budget is Algorithm 1 with M nodes per pass: in every round, it expands the heaviest open prefixes that fit, one level per round, and it stops when the certificate holds or when all Y passes have been used. We call this run the *unplanned run*. It never fails, in the sense that, by Proposition 2, the list that it holds at the end is within the factor $1 + \mathrm{gap}$ at every rank. However, when Y passes are too few for the memory M, that list is not exact, its factor is whatever gap turns out to be, and nothing about this factor is known before the last pass.

In contrast, a factor that is stated after the first pass has to hold against the adversary, so the remaining passes must cover the worst case. In particular, if the remaining rounds cannot resolve a heavy open prefix one level at a time, its skeleton has to be fed now, before the model has revealed whether it is needed. Thus, the two runs differ only in line 7 of Algorithm 2. Both expand the prefixes that are known to be open, in the same order, as long as they fit. In addition, BUDGET-SIDECAR feeds the speculative nodes. It first places them in the slots that the open prefixes leave empty, so that no open prefix is delayed. Only when the schedule requires more speculative nodes than there are empty slots do they take the place of the lightest open prefixes outside the pool, which then wait for a later round.

Because the speculative nodes change what is known after each round, the two runs do not expand identical nodes in later rounds. Nevertheless, the lists that they return within a fixed budget differ little, because the speculative nodes are what the adversary would need, and they rarely hold an item that the real model needs. The planning adds two things. First, the deployment knows the factor and $Y_{\min}$ after the first pass. Second, when the passes are not capped, the certificate arrives sooner, since feeding a skeleton whole resolves a cluster in one round, where the level-by-level fallback needs several. The difference is sharpest in the worst case, which the unplanned run does not cover. When a heavy cluster hides the items of MODEL-K across its skeleton, the level-by-level descent reaches those items only one level per round, so it can exhaust a budget that suffices for exactness, whereas BUDGET-SIDECAR feeds the skeleton whole and is certified within the same budget, as the following proposition states.

Proposition 5 (worst-case planning is necessary)**.** *There exist a catalog trie, a draft, a model and a budget (M, Y) with $Y < L$ under which the unplanned run, i.e., SIDECAR (Algorithm 1) with M nodes per pass, returns a list that contains no item of* MODEL-K*, whereas BUDGET-SIDECAR announces exactness after the first pass and returns* MODEL-K *with its certificate within the same budget.*

The instance in the proof is the worst case for the unplanned run, i.e., a model whose probability sits in one heavy cluster and is spread evenly over the items below it. On this instance, only the planning lets the budget suffice for exactness.

Theorem 2 (announced factor under a budget)**.** *Run BUDGET-SIDECAR with M nodes per pass for at most Y passes. Then:*

(i) SIDECAR-K *is within a factor $1 + \delta^*(M, Y)$ of* MODEL-K *at every rank;*

(ii) $Y_{\min}(M) = 1 + R^*(s_K(0))$ *passes certify* SIDECAR-K*, a number known after the first pass;*

(iii) $\delta_{r+1}^* \le \delta_r^*$ *for every round r: re-planning never loosens the announcement.*

Remark 1 (Cost of the worst-case guarantee.). The announced factor holds against the adversary, including a completion that spreads the probability of a heavy cluster across its whole skeleton. A real model, however, concentrates that probability on a few items, often on ones that the first pass has already scored. Thus, the realized gap is usually far below the announced factor, and a deployment reports both. In other words, the announcement is pessimistic, whereas

the planned run is not. Apart from the speculative nodes of line 7 of Algorithm 2, the planned run feeds only what every exact decoder must feed.

The speculative nodes are therefore the only cost of the announcement. When the schedule forces them into a full batch, they replace prefixes that the real model may need, and since under the real model most of them turn out to hold no item of MODEL-K, the overlap of the planned run with MODEL-K can end a few percent below that of the unplanned run with the same budget. When the announced factor is far above one, the useful output is the announcement itself, since it tells the deployment after one call that its budget is inadequate. Moreover, Eq. (5) tells the deployment by how much, since for the given memory M it gives the number of passes that exactness requires, and for the given number of passes Y it gives the memory that exactness requires.

Remark 2 (Deployment parameters.). BUDGET-SIDECAR has two budgets and one target. The budgets are M and Y, and for $Y = L$ and $M = W$ they are the resources of beam search of width W. Everything else in BUDGET-SIDECAR follows from the budgets, the first pass, and the catalog trie, so there is no parameter to tune. The target δ_t states what quality is sufficient, and it can only shorten the planned run: line 4 of Algorithm 2 returns as soon as SIDECAR-K is certified within $1 + \delta_t$, and with $\delta_t = 0$ it returns at exactness. Note that this stopping rule rests on the certificate alone and needs no planning.

What the planning adds is that the deployment knows after the first call whether the target will be met within the budget, which is the case exactly when $\delta_t \geq \delta^*(M, Y)$, and how many passes exactness would take. Hence, the deployment can serve, escalate, or commit to a latency before the fallback runs. The budget and the target, however, do not play symmetric roles, since Y binds unconditionally, whereas the target only allows the planned run to stop early. Hence, the guarantee is $\max(\delta^*(M, Y), \delta_t)$: if $\delta_t \geq \delta^*$, the budget can deliver the target and the planned run may end early, and if $\delta_t < \delta^*$, it uses all Y passes and SIDECAR-K is certified within $1 + \delta^*$, which the deployment learns after the first pass. Finally, the draft is an input rather than a parameter to tune. It determines how small δ^* is for given M and Y, but it never affects whether SIDECAR-K and its certificate are correct.

5. Experiments

We now evaluate SIDECAR and BUDGET-SIDECAR against constrained beam search. First, we describe the dataset, the model, and the replay procedure that allows us to validate our claims on every request. Then, we introduce the baseline and the draft sources. Finally, we state the research questions of interest to a practitioner and answer them in the rest of the section.

Data and model. For the purpose of this paper, we use the Beauty category of Amazon Reviews 2014 (McAuley et al., 2015; He & McAuley, 2016), which is commonly used to evaluate SID-based generative retrieval approaches. The dataset has 22,363 users and 12,101 items, and we split it with the leave-one-out protocol. The SIDs come from an RQ-VAE with three codebooks of 256 codes each, followed by a collision code, so that every SID has $L = 4$ codes. We train a four-layer decoder-only transformer on the histories of the users, written as sequences of SID codes. It should be noted that SIDECAR only modifies the decoder stage, and therefore we do not focus on optimizing the hyperparameters of the transformer to obtain the most performant model. However, the ranking quality of our trained model is within the confidence intervals of the published works using this setup.

Evaluation protocol. For every test user, we run one tree pass over all 21,091 internal nodes of the catalog trie to obtain the exact distribution of the model's probability mass. Every decoder is then a simple deterministic replay of these stored distributions. Thus, we can test every claim on every request, and compare the decoders on identical scores. We use MODEL-K with $K = 10$ as the reference target, since it is the output of exhaustive scoring of the model. The goal of SIDECAR is to return the output of exhaustive scoring; no decoder can do better than the model that it decodes. Thus, every comparison in this section uses MODEL-K as the reference for both decoders.

Flatter models. In order to vary the sharpness of the model without changing the decoders, we apply a temperature T to every conditional distribution of the trained model. The trained model corresponds to $T = 1$, and a larger T gives a flatter model. Note that a temperature has a different effect here than on a single softmax that is commonly used in large language models. For a single softmax, a temperature rescales the distribution but does not change the order of its outcomes. Here, however, each of the L conditionals is tempered at its own prefix and renormalized over the children of that prefix, so the ranking of the items changes. As T grows, the probability of an item tends to the product of one over the number of children of each prefix on its path. The flatter models therefore rank the items differently from the trained model and serve as a useful proxy for a model trained on the noisy data that is common in large-scale recommender systems, since the conditionals of such a model are similarly spread out.

Baseline. Our baseline is constrained beam search with the same model and the same history cache, i.e., the setup used by TIGER. We do not compare against the speculative

decoders for SIDs, because each of them requires an artifact beyond the trained model. AtSpeed (Lin et al., 2025) requires a trained drafter, NEZHA (Wang et al., 2026b) requires a trained self-drafting head, and SpecGR (Ding et al., 2026) requires a tuned acceptance threshold. Moreover, these decoders aim to reproduce the list of beam search in fewer passes, and none of them certifies it. Hence, BEAM-K is also the reference against which these decoders are measured.

Potential draft sources. At request time, SIDECAR queries drafts from key-value stores. Such drafts come predominantly from two kinds of sources, namely model-free and model-based sources. The *model-free* sources do not use the SID model at all. Content kNN returns the items whose content embeddings are closest to any of the last five items of the history. Co-occurrence and matrix factorization (implicit ALS) are built from the interaction log. The *model-based* sources, in contrast, derive the draft from the SID model, and the main one is the *user sequence*, i.e., the top-N list of the model for the user, which a batch job computes and stores. Note that, throughout this section, the user sequence does not denote the items in the history of the user. Instead, it denotes the N items that the SID model itself ranks highest when a batch job runs it on the stale history of the user. We consider three variants of the user sequence. The first variant is the sequence stored by a copy of the model whose weights are quantized to FP4, which a cheaper batch job could run using constrained beam search. The other two variants are copies of the sequence that are two and five interactions old. Irrespective of the source, the drafts are always older than the request that triggered decoding. RQ2 compares all seven sources, and from RQ4 onwards, in the interest of brevity, every figure uses two drafts of 500 entries each. The first draft is the user sequence, which is our default. The second draft comes from matrix factorization, the strongest model-free source, which we henceforth refer to as the MF draft.

What the experiments measure. We checked every property proven in Sections 2 to 4 on every request of every experiment to ensure that none was violated. Specifically, no certified list differed from MODEL-K, no announced factor was exceeded, and no run took more passes than its worst-case bound. Since correctness is thus settled, the following experiments focus on analyzing costs. They address seven research questions (RQs), the first four with unlimited memory per pass and the other three under the budgets of beam search:

RQ1. What decides the number of passes?
RQ2. What is a good source of drafts?
RQ3. How does the size K of the output list affect the costs?

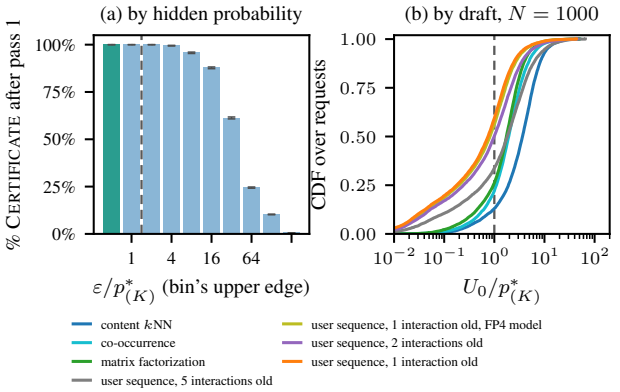


Figure 4. The hidden-probability condition. (a) Left of the line, the first pass is certified on every request, as proven, and it still is on most requests up to eight times the threshold. **(b)** The exact condition, $U_0 < p^*_{(K)}$, orders the sources as their passes do.

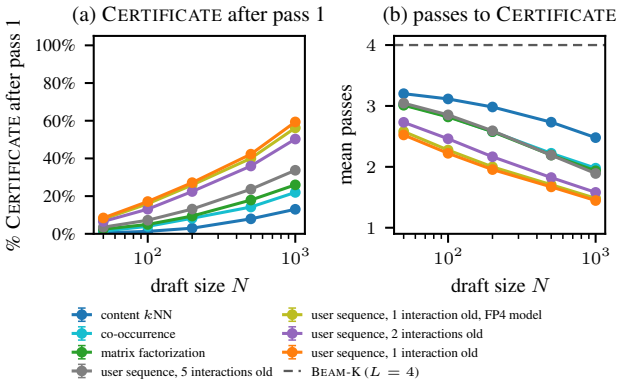


Figure 5. The draft sources. Every source yields an exact list, and the passes follow its alignment with the model: the model's own stale ranking needs well under two passes, even from an FP4 copy of the model, the behavioral sources about 2.2, and content similarity close to three. Beam search always needs four passes.

RQ4. What incremental benefit does the catalog trie provide?
RQ5. At beam search's resources, is SIDECAR-K at least as good as BEAM-K?
RQ6. What does exactness of SIDECAR-K cost?
RQ7. What benefit does the worst-case guarantee of BUDGET-SIDECAR provide, and what does it cost?

Finally, we present the overall landscape of the results on one budget plane.

Reporting in the figures. Every figure in this section reports a statistic over requests, with one request per user. The text that discusses each figure provides the context of its experiment, and wherever a setting is not mentioned, the figure uses 10,000 held-out users, drafts of $N = 500$ entries, and $K = 10$. The error bars in the figures show 95% confidence intervals over these requests, and they are often so narrow that they are hardly visible.

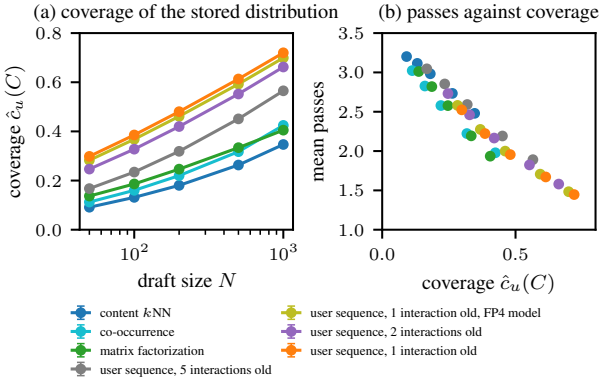


Figure 6. Alignment of the draft sources, measured offline. (a) The offline coverage $\hat{c}_u(C) = \sum_{e \in C} \hat{p}_u(e)$ of each draft against its size. **(b)** The passes of SIDECAR against the offline coverage, one point per source and size. All sources fall on one curve, so the offline coverage of a source predicts its number of passes and can be used to choose among sources.

5.1. Number of Passes (RQ1)

In this section, we address RQ1: what decides the number of passes? Intuitively, the number of passes should depend on how much of the probability of the model the draft leaves uncovered. The reason is that, by Section 2, the first pass is certified whenever this hidden probability is below the probability of the K-th best item.

In order to test this intuition, we group the requests by the ratio of the hidden probability to the probability of the K-th best item. We use unlimited memory per pass, so that the probability mass of the draft is the only factor that matters. In Fig. 4(a), we observe that the first pass is certified on every request to the left of the threshold, as proven. Interestingly, the certificate still holds on most requests whose ratio is up to $8\times$ the threshold. This happens because the hidden probability is rarely concentrated in one place. Instead, it is spread over many uncovered prefixes, and each of these prefixes is lighter than the K-th best item. Hence, the sufficient condition of Section 2 is far from necessary.

Panel (b) shows the exactness condition, which compares the heaviest frontier prefix U_0 with the K-th best item, as a distribution over requests for each source at $N = 1000$. The first pass is certified exactly on the requests to the left of one. The sources whose drafts leave less probability uncovered place more of their requests to the left of this line: the curve of the user sequence lies furthest to the left, and the curve of content kNN furthest to the right. These observations indicate that whenever the draft misses a cluster, the fallback needs at least one more round, because every level-1 prefix that the draft leaves uncovered has to be either opened or ruled out. Thus, we conclude that the probability that the draft leaves uncovered, and in particular its heaviest uncovered prefix, decides the number of passes.

5.2. Draft Sources (RQ2)

Next, we address RQ2: what is a good source of drafts? Since every source yields MODEL-K, the choice of source only affects the number of passes. We find that the sources that are aligned with the SID model need the fewest passes, and that this alignment can be measured offline, before deployment.

Figure 5 compares the seven sources with unlimited memory per pass. Broadly, their costs fall into three separate tiers. Content kNN needs close to three passes at any draft size, and its first pass is certified on fewer than 10% of requests at 500 entries. The two behavioral sources, which read the interaction log, need about 2.2 passes at 500 entries, and matrix factorization is slightly better than co-occurrence. A user sequence that is one interaction old needs 1.7 passes, and its first pass is certified on about 40% of requests. The sequence stored by the FP4-quantized copy of the model needs only three hundredths of a pass more, although the weights of that copy are 11% off and its top-10 agrees with the top-10 of the full model on only 69% of requests. A user sequence that is two interactions old, in contrast, needs another sixth of a pass. A user sequence that is five interactions old needs 2.2 passes, which is the same as the MF draft, although it still leaves less probability uncovered and is certified after one pass more often. Thus, SIDECAR tolerates both a cheaper quantized model and a stale history.

One may expect recall against the held-out item to explain this order. However, the MF draft has nearly the same recall as the user sequence, yet it needs half a pass more. Instead, the alignment of the draft with the model explains the order. The behavioral sources contain items from the right clusters more often than content similarity does, which is why they need half a pass less than content kNN. However, they rarely contain the top items of the model itself.

Moreover, this alignment can be measured before deployment. Let $\hat{p}_u$ denote the distribution of the model for user u one interaction ago. The user sequence is the top-N of this distribution, so the batch job that produces the user sequence has already computed $\hat{p}_u$. We define the *offline coverage* of a draft C as the stored probability of its items, i.e., $\hat{c}_u(C) = \sum_{e \in C} \hat{p}_u(e)$. A batch job can read this quantity off the stored table without running the decoder. The offline coverage is the counterpart of $1 - \varepsilon$ for the model based on stale user history, computed over the items of the draft alone. In Fig. 6(a), we observe that the offline coverage ranks the sources exactly as their numbers of passes do. Panel (b) plots the mean number of passes against the offline coverage, and every source at every size lies on a single curve. This confirms the conclusion of RQ1 with a quantity that can be computed offline. Thus, at the time of deployment, one can determine the best source and the size of the draft by simply choosing the source whose offline

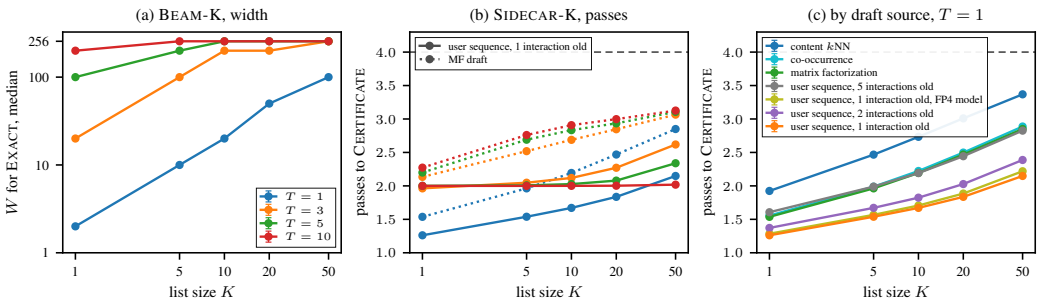


Figure 7. The size of the list. **(a)** The beam width for an exact list grows with K on the trained model and sits at the largest width run on the flatter ones. **(b)** SIDECAR's passes rise gently with K on the trained model, and with the user sequence they stay at two on the flat models whatever K is. **(c)** The order of the draft sources holds at every K.

coverage is highest at the size permitted by the memory budget of the deployment.

5.3. Size of the List (RQ3)

Next, we address RQ3: how does the size K of the output list affect the costs? Like the sharpness of the model, the size of the list should be determined by the application rather than by the decoder. We expect a longer list to make exactness harder, since the K-th best item is lighter and more prefixes can potentially outweigh it. In order to quantify this effect, Fig. 7 varies K from 1 to 50 with unlimited memory per pass.

As expected, both decoders need more resources for a longer list, but they need different resources. Beam search needs a wider beam (panel a). On the trained model, the median request needs a width of 2 for the single best item and a width of 100 for the top-50. On the flatter models, the median request needs the largest width that we run from $K = 5$ on. SIDECAR, in contrast, needs more passes, and only a few more (panel b). With the user sequence on the trained model, the number of passes rises from 1.2 to 2.2 across the same range of K. On the flatter models, it stays at two for every K, because the first pass already holds the list and one fallback round certifies it, whatever its length. Panel (c) shows that the order of the sources found in RQ2 holds at every list size, with the MF draft about half a pass behind the user sequence. Thus, the size of the list affects the width of beam search by up to 50×, whereas it only changes the number of passes of SIDECAR by about one and still requires fewer passes than beam search for $L = 4$.

5.4. Benefit of the Catalog Trie (RQ4)

In this section, we address RQ4: what incremental benefit does the catalog trie provide? In short, the structure of the catalog trie allows us to allocate tokens, i.e., memory, efficiently in the first pass, which can reduce the number of fallback passes required. Two properties of the catalog trie make this possible. First, the content codes address 16.7 million cells for only 12,101 items. As a result, 78% of the depth-2 prefixes and 98% of the depth-3 prefixes hold a single item, and two thirds of the candidate prefixes of every draft lie on chains. Second, the skeleton of the catalog consists of 2,533 of the 21,091 internal nodes of the trie, i.e., of about ten nodes per level-1 cluster.

Figure 8 quantifies the effects of these two properties. In panel (a), we observe that the chain rules of Section 3 cut the first pass to a third of its tokens at every size, while the scores stay identical. Panels (b) and (c) compare the two ways of resolving an open cluster, namely level by level and with its whole skeleton. The skeleton round never needs a third pass if enough memory is available. It reduces the passes of the MF draft from 2.2 to 1.8 on the trained model and from 2.9 to 2.0 on the flattest model. However, the skeleton round feeds about 7× as many fallback tokens to achieve this reduction. Moreover, it saves the user sequence little, because the chain rules alone already keep that draft near two passes. Thus, feeding a skeleton whole saves a pass only when the draft leaves heavy clusters open, but it comes at the cost of additional tokens in fallback rounds.

5.5. Equal Resource Comparison (RQ5)

Building upon these results with unlimited memory, we now compare the two decoders at equal resources and address RQ5: is SIDECAR-K at least as good as BEAM-K? In Fig. 9, beam search of width W runs with its own resources, namely four passes with W prefixes each. BUDGET-SIDECAR receives the same resources, namely $M = W$ nodes per pass and at most four passes, and all its passes, including the first pass, are capped at M nodes. We find that SIDECAR-K is at least as good as BEAM-K at every budget, and much better where the model is flat. Moreover, at every budget, SIDECAR states how close its list is to MODEL-K, either with a certificate or with the announced factor, whereas beam search gives no indication of how far its list is from the exact one.

At beam search's latency. The top row of Fig. 9 fixes the number of passes at four and varies the memory per pass. On the trained model, the black and the green bars

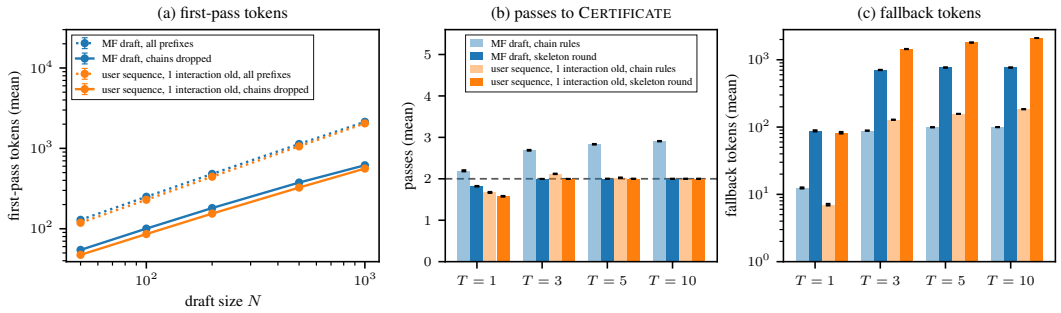


Figure 8. What the catalog trie saves. (a) Dropping the chain prefixes cuts the first pass to a third of its tokens. **(b, c)** The skeleton round never needs a third pass, but it saves up to one pass only for the MF draft, and it feeds several times as many fallback tokens.

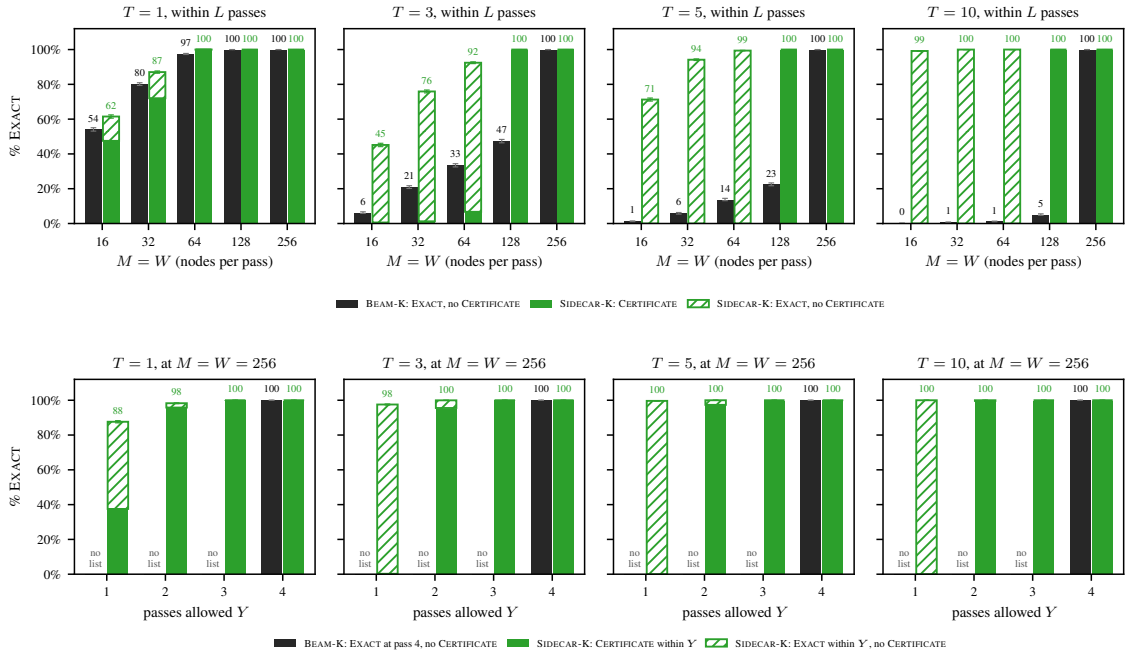


Figure 9. Equal resources against beam search. Top: beam search's latency of four passes, against memory. **Bottom:** beam search's memory of 256 nodes, against passes. On the sharp model the lists agree and only one decoder knows it. On the flat models SIDECAR-K is exact at a fraction of the memory and two passes earlier.

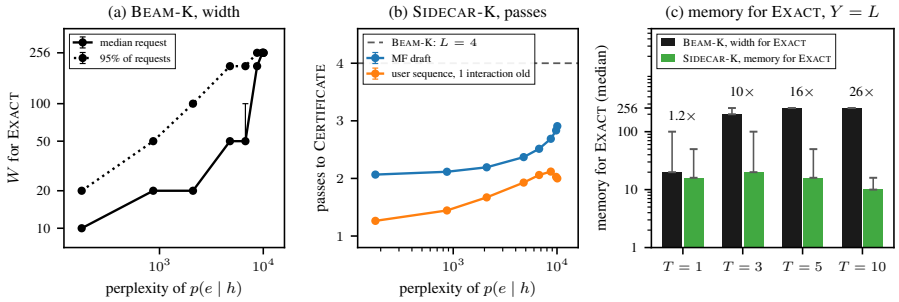


Figure 10. What exactness costs as the model flattens. Beam search needs a beam width that it cannot determine in advance (a), whereas SIDECAR needs a bounded number of passes (b). On the memory axis (c) the gap opens to more than an order of magnitude on the flat models.

rise together and reach every request at the same memory. Thus, BEAM-K and SIDECAR-K coincide, and a deployment that looked only at the lists would see no difference. The difference, however, lies in the solid part of the green bars, i.e., in the share of requests on which SIDECAR has proven its list exact. From a modest memory on, this share covers nearly all requests. Beam search, in contrast, has no such share at any width, because nothing in its computation distinguishes an exact list from a wrong one. On the flat models, correctness itself separates the two decoders. SIDECAR-K is exact on most requests at the smallest memory and on every request at a fraction of beam search's width, whereas BEAM-K needs the full width to become exact. It should be noted that at small memories, SIDECAR returns the exact list before it can prove that the list is exact, as the hatched part of the green bars shows. The reason is that the open clusters cannot all be closed within the three fallback rounds.

At beam search's memory. The bottom row of Fig. 9 fixes the memory per pass at beam search's width and varies the number of passes. After a single pass, SIDECAR already holds MODEL-K on most requests of the sharp model and on nearly all requests of the flat models. At that point, beam search does not yet hold any list, since it needs three more passes to output a list. One pass later, SIDECAR has certified nearly all of its lists. Beam search has a single bar, at its fourth pass, where its lists are exact on these requests at this width. Hence, a deployment that serves the list of SIDECAR after two passes compares favorably against one that waits for the four passes of beam search.

Why flatness helps SIDECAR-K and hurts beam search. It may seem counterintuitive that the performance of BUDGET-SIDECAR improves with the flatness of the underlying model. Here, we explain the reasoning behind it. This pattern follows from what determines the top-K of a flat model. The probability of an item is a product of L conditionals. As the model flattens, each conditional tends to one over the number of admissible children, so the probability of an item tends to the product of these reciprocal branching factors along its path. The top-K of a flat model then consists of the items in small subtrees, which share the probability of their cluster with few siblings, regardless of the history. Indeed, on the trained model, the clusters that hold the top-10 are among the heaviest clusters of the trie. On the flattest model, in contrast, these clusters have median probability mass, and all of them are below the median size. This mechanism has three manifestations. First, the user sequence of a flat model, which is its ranking for the stale history, almost surely contains the top items for the current history, because the ranking is now determined by the shape of the catalog rather than by the history. Thus, staleness matters less and less as the model flattens, and the first pass already returns the exact list. Second, beam search ranks prefixes by their probability mass. It therefore keeps the heavy clusters and drops the light ones, although the light clusters are the ones that hold the top items. This is why the bars of beam search collapse on the flat models until its width covers every cluster. Third, the certificate has to rule out the heavy clusters. Each frontier prefix of a heavy cluster outweighs the small top items, even though none of these prefixes holds one of them, and ruling these prefixes out takes rounds. In other words, flatness separates where the probability mass is from where the best items are located in the trie.

5.6. Cost of Exactness (RQ6)

Next, we address RQ6: what does exactness of SIDECAR-K cost? Figure 10 varies the sharpness of the model and reports the resources required by each decoder to output an exact list. We find that on a sharp model the scarce resource is the number of passes, whereas on a flat model it is the memory per pass. On both kinds of model, SIDECAR needs less of the resource that is scarce.

Beam search needs width, i.e., memory for a wider beam (panel a). A flat model spreads its probability over many prefixes of nearly equal mass, so the beam has to be wide enough to keep the right ones. Across the sweep of temperatures, the width that the median request needs grows more than 10×. From a moderate flatness on, the width that 95% of requests need reaches the largest width that we test. Moreover, the width required to achieve exactness for beam search is a property of the request that beam search learns only in hindsight, so a deployment has to provision every request assuming the worst-case scenario. SIDECAR, in contrast, needs extra passes to achieve exactness (panel b), and this cost is bounded by construction. With the user sequence, it levels off at two passes on the flat models, and with the MF draft, it rises toward three.

Panel (c) compares the two decoders on the memory axis, since memory per pass determines how many requests a serving machine can process at once, and hence the size of the serving fleet. On the trained model, both decoders need about the same memory for an exact list. There, SIDECAR gains in passes, since at equal memory it needs fewer than half of beam search's passes. As the model flattens, however, the memory that beam search needs grows, whereas the memory that SIDECAR needs does not. The gap opens to more than an order of magnitude, and at the flat end it is 10–26×. This is a large difference in favor of BUDGET-SIDECAR that affects the practicality of the two decoders.

Figure 12 shows both resources in one plane. Beam search is a single point in the plane of memory and passes, at four passes and the width that the median request needs. BUDGET-SIDECAR, in contrast, is a curve, since more

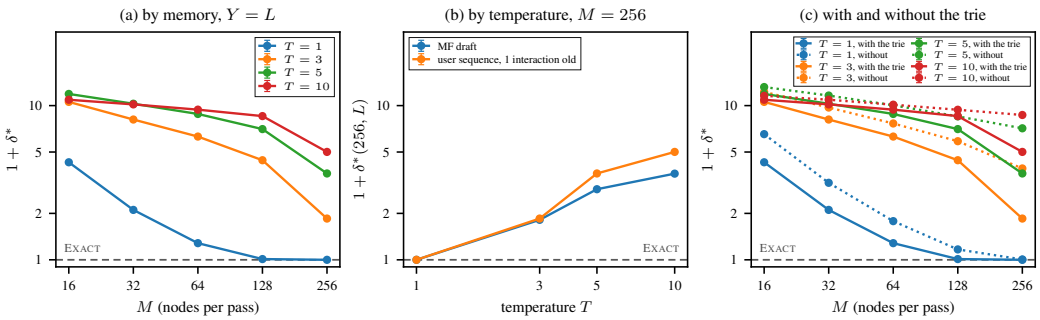


Figure 11. The announced factor, against memory at four passes (a) and against temperature at beam search's memory (b). It rises as the model flattens for both drafts, and at beam search's memory it is lower for the MF draft, whose candidates spread over more clusters, although that draft is exact far less often at smaller memories. The realized gap was zero at every budget. **(c)** With the catalog trie (solid) the factor is never looser than the count-based announcement without it (dotted, Eq. (7)), and it is about half of it on the flat models at beam search's width.

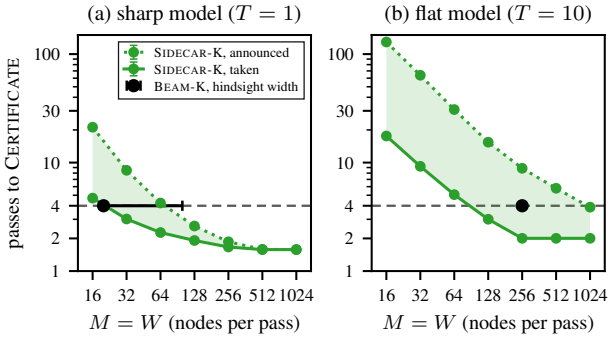


Figure 12. Passes against memory for a certified list. Beam search is a point at its four passes, and BUDGET-SIDECAR is a curve below and to its left on both models.

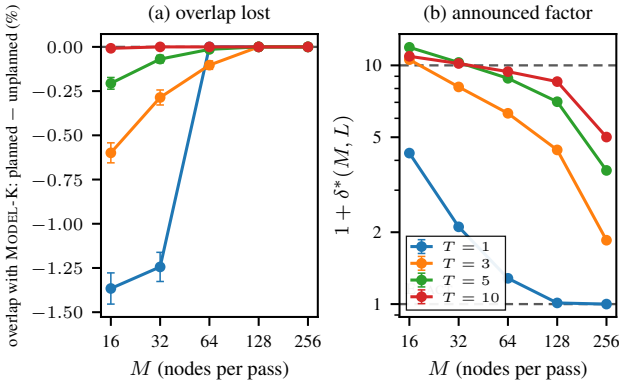


Figure 13. The price of the announcement, user sequence. **(a)** The overlap of BUDGET-SIDECAR's list with MODEL-K minus that of the unplanned run, in percent. **(b)** The announced factor at the same budgets. The price is at most one or two percent, and only at the smallest memories.

memory per pass reduces the number of passes to a certified list. On both models, the curve lies below and to the left of the point. Hence, both a deployment that is short of memory and a deployment that is short of latency gain from SIDECAR.

5.7. Cost vs. Benefit of the Worst-Case Guarantee (RQ7)

Having measured what the planned run achieves, we finally address RQ7: what benefit does the worst-case guarantee of BUDGET-SIDECAR provide, and what does it cost? The guarantee is the announced factor $1 + \delta^*$ of Theorem 2, which holds within the budget against the adversary described in Section 4. An unplanned run has no such guarantee: until its last pass, it cannot say how far its list is from MODEL-K. In the worst case, it returns no item of MODEL-K at all, even though the same budget suffices for exactness when the run is planned (Proposition 5). We find that the announced factor reaches exactness within beam search's budget on the trained model, and that the guarantee lowers the overlap with MODEL-K only at budgets where the factor is too large to be useful.

Size of the announced factor. Figure 11 shows the announced factor in three panels. In panel (a), the number of passes is fixed at four, as for beam search, and the memory per pass varies. On the trained model, the factor drops to one at beam search's width. A deployment with that budget therefore knows after the first pass that its list will be MODEL-K within four passes. On the flat models, the factor falls more slowly with memory, because every open cluster carries more probability and the adversary may hide a top item anywhere below it.

In panel (b), the memory per pass is fixed at beam search's width and the temperature varies. The factor rises as the model flattens. It is lower for the MF draft than for the user sequence, because a behavioral draft spreads its 500 candidates over more clusters and leaves less probability uncovered. Exactness, however, ranks the two drafts the other way, as shown in RQ5. Thus, the announcement measures how much probability a draft leaves uncovered, whereas exactness depends on whether the draft contains the top items. For the user sequence, the realized gap was

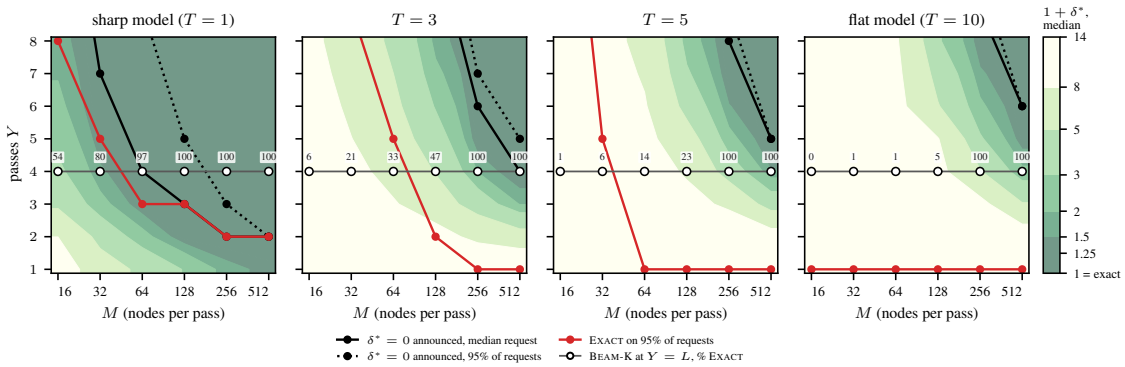


Figure 14. The budget map. Color: the factor announced after the first pass at each budget, darkest for exactness. Black: the smallest Y at which exactness is announced, for the median request (solid) and for 95% of requests (dotted). Red: the smallest Y at which SIDECAR-K is exact on 95% of requests. Beam search is the row $Y = L$ with its share of exact lists. The announced and the realized frontier coincide on the sharp model and part ways on the flat one.

zero at every one of these budgets, so only the certification fell short.

Panel (c) compares the announcement of Theorem 2 with an announcement that uses only the number of items below each prefix (Proposition 8 in Appendix B). Including the catalog trie never loosens the factor, and it roughly halves it on the flat models at beam search's width. The reason is that, without the trie, the announcement must assume that every open cluster is resolved level by level, which can take a node for every item below it. With the trie, a cluster can instead be resolved by feeding its whole skeleton in one round, which takes only about ten nodes. This is a further benefit of the trie, in addition to the chain rules and skeleton rounds of RQ4.

Overlap lost to the guarantee. To guarantee the worst case, BUDGET-SIDECAR uses some of the nodes in each fallback pass for speculative prefixes, which the real model may not need. The unplanned run uses all of its nodes for prefixes that are known to be open, so its list can overlap more with MODEL-K. Figure 13(a) shows this difference. For the user sequence, the two runs overlap equally from a moderate memory on, and BUDGET-SIDECAR loses 1–2% of overlap below that memory. The loss occurs only at small memories, where the announced factor is large (Fig. 13(b)). Even there, the announcement tells the deployment after the first pass that its memory is too small, and Eq. (5) gives the number of fallback rounds that it would need to certify its list. It should be noted that whether speculation helps or hurts the overlap depends on the model as much as on the procedure. The speculative nodes are the nodes that the adversary would need. Hence, they improve the list when heavy clusters hide items of MODEL-K, and they slightly lower the overlap when the heavy clusters hold no such items. On this benchmark, the top items of a flat model lie in light clusters (RQ5), which is the second case. Hence, a deployment should announce the factor, which is the commitment that it can make before the fallback runs, and report the realized gap afterwards, which can only be smaller.

5.8. Overall Landscape

Figure 14 summarizes the section on a single visual plane. Every point of the plane is a budget of M nodes per pass and Y passes. The color at a point is the factor that BUDGET-SIDECAR announces at that budget after its first pass. Thus, for any budget set by a deployment, the map shows what BUDGET-SIDECAR can promise before the fallback runs. The map shows two frontiers. The dashed curve marks where SIDECAR-K is exact on nearly every request, as measured by RQ5 and RQ6. The solid curve marks where this exactness is also announced after one call, which is the guarantee of RQ7. Beam search is the row $Y = L$, and it announces nothing at any budget.

On the sharp model, the two frontiers nearly coincide, and both cross the row of beam search at a fraction of its required width. Thus, at budgets where BEAM-K is merely exact, SIDECAR-K is exact, announced as exact, and obtained in a fraction of the passes. On the flat model, the two frontiers diverge. The realized frontier drops to a single pass at every memory shown, whereas the announced frontier retreats to the top right corner. As a result, at beam search's own budget, the guarantee is in the form of the announced factor rather than exactness. Between the two frontiers, SIDECAR-K is exact, but the announcement cannot yet say so. This region is wide on the flat model because the real model concentrates probability where the draft already looks, whereas the announcement must allow for the worst case, in which the adversary spreads that probability elsewhere.

The map thus separates the two decisions that a practitioner has to make at the time of deployment. Above the dashed curve, the deployment can expect an exact list. Above the solid curve, it can also commit to exactness after one call. Between the two curves, it receives SIDECAR-K together

with a factor that it can state. In practical terms, when the model is sharp, BUDGET-SIDECAR is preferable outright, since it returns the list of BEAM-K in a fraction of the passes and certifies it. When the model is flat, memory is the deciding resource, and SIDECAR-K is exact with a fraction of the memory of beam search or in half of its passes. When neither resource suffices for exactness, BUDGET-SIDECAR still states how close SIDECAR-K is to MODEL-K.

Early stopping. The certificate and the announced factor also allow a deployment to stop early, in three ways that beam search cannot offer. First, the decoder stops as soon as its list is certified. The bottom row of Fig. 9 shows that, at beam search's memory, nearly all lists are certified after two passes, so the remaining passes of the budget are not spent. Second, a deployment that accepts an approximate list can set a target factor and stop as soon as the list is certified within it. The colors of Fig. 14 show, for every budget, the factor that is announced after the first call on the median request, and hence the budgets at which a given target is met. Third, a deployment with a latency deadline can stop after any pass. The bottom row of Fig. 9 shows that the list after a single pass is already exact on most requests of the sharp model and on nearly all requests of the flat models, and the dotted curve of Fig. 12 shows after the first call how many passes the certificate will need. In all three cases, the deployment can also act on the announcement itself: when the announced factor after the first call exceeds its target, it can stop at once and serve the list with its stated factor, or escalate the request to a larger budget.

6. Related work

To the best of our knowledge, SIDECAR is the first decoder for Semantic IDs that proves its list to be the exact top-K of the model or, under a budget, states a factor for its list before the fallback runs. Our work connects to several streams of research, which we discuss below.

Generative retrieval and Semantic IDs. One line of research frames retrieval as the generation of document identifiers: the Differentiable Search Index (Tay et al., 2022) introduced the idea, and the Neural Corpus Indexer (Wang et al., 2022) and query-generation variants (Zhuang et al., 2022) developed it for documents. Subsequent work studied its limits at corpus scale (Pradeep et al., 2023), its reading as dense retrieval over prefixes (Nguyen & Yates, 2023), and ranking objectives over identifiers (Li et al., 2024b; Zeng et al., 2024). TIGER (Rajput et al., 2023) brought Semantic IDs to recommendation with codes from an RQ-VAE (Lee et al., 2022), in the manner of residual quantization in neural codecs (Zeghidour et al., 2022), and decoded them with constrained beam search. Item identifiers for language-model recommenders were studied by Geng et al. (2022); Hua et al. (2023). Another line of research learns the tokenizer from collaborative signals (Jin et al., 2024; Zheng et al., 2024a; Wang et al., 2024a; Zhu et al., 2024; Liu et al., 2025; Qu et al., 2025; Si et al., 2024; Wang et al., 2024b). Moreover, Semantic IDs were shown to improve generalization in ranking models (Singh et al., 2024). The depth of the SID tree is still an open choice: an extra level can improve recommendation up to a point (Fang et al., 2026), a shallower SID has improved offline and online metrics (Xie et al., 2026), and a deeper TIGER degraded under beam search (Hou et al., 2025), although depth changes the serving cost and the model at the same time. Industrial systems, such as PLUM (He et al., 2026), GLIDE (D'Amico et al., 2026), OneRec (Deng et al., 2025; Zhou et al., 2025), COBRA (Yang et al., 2025), and the generative recommenders of Meta (Zhai et al., 2024; Yang et al., 2024), all decode with beam search over the SID trie, and a practitioner's study catalogs their design choices (Ju et al., 2025). Although these works improve what the model ranks, they are orthogonal to ours, as they leave the decoding of the ranking unchanged. In comparison, we leave the model and its tokenizer unchanged and change only the decoder. Our approach applies to any of these tokenizers.

Beam search and its alternatives. Oh et al. (2026) showed that a single head with beam search gives better ranking and content quality than multi-token prediction (Gloeckle et al., 2024), which parallel SID decoders (Hou et al., 2025; Zhan et al., 2026) adopt for throughput. Wu et al. (2025) showed that constrained beam search can reach high top-1 precision while recalling branches poorly at top-k. Before Semantic IDs, tree-based deep models (Zhu et al., 2018; 2019) and Deep Retrieval (Gao et al., 2021) retrieved with beam search over learned node scores, in the lineage of hierarchical softmax (Morin & Bengio, 2005), and Zhuo et al. (2020) showed that beam search over such trees is not Bayes-optimal and trained the scorers to compensate. These works adapt the model to the decoder; in contrast, we take the opposite route, in which the model stays as trained and the decoder becomes exact and certified.

Constrained decoding and exact search. Constrained decoding limits each next token to the choices that are allowed. GENRE (De Cao et al., 2021) did so with a trie of entity names, and PICARD and grammar-constrained decoding did so with a grammar (Scholak et al., 2021; Geng et al., 2023). Decoding Semantic IDs is the simplest such case, because every item is a path of the same length in a trie, and the probability of a prefix is the total probability of the items below it. Hence, a prefix is never more likely than its own parent; exact search methods rely on this fact. These methods include exact search for the single best sequence (Stahlberg & Byrne, 2019) and best-first

beam search (Meister et al., 2020b;a). However, they search for one sequence of unbounded length, where exactness is expensive. In contrast, we search for the top-K list over a trie of fixed depth L, where one pass over a good draft often suffices to certify the list, and L passes always do. A similar idea appears in classical retrieval, where WAND and block-max WAND (Broder et al., 2003; Ding & Suel, 2011) certify a top-K list using score upper bounds that are computed offline. In contrast, our upper bounds come from the same forward pass that scores the draft, and the fallback expands only the prefixes that violate them.

Speculative decoding. Speculative decoding (Leviathan et al., 2023; Chen et al., 2023; Sun et al., 2023) drafts tokens cheaply and checks them against the target in one pass. Tree attention extends this to a whole draft tree (Miao et al., 2024; Cai et al., 2024), and subsequent work added draft heads on the target (Ankner et al., 2024; Li et al., 2024a), lookahead and self-speculation (Fu et al., 2024; Zhang et al., 2024), staged and big-little drafters (Spector & Ré, 2023; Kim et al., 2023), retrieval-based drafts (He et al., 2024), and draft trees chosen under a token budget (Chen et al., 2024). Perhaps closest to our objective are the speculative decoders for generative recommendation: AtSpeed (Lin et al., 2025) adapts speculative decoding to beam search and trains an aligned drafter, NEZHA (Wang et al., 2026b) adds a self-drafting head, SpecGR (Ding et al., 2026) drafts by retrieval and accepts by a likelihood threshold, and STARK (Sun et al., 2026) shares the KV cache across beams. These decoders accelerate generation, but none of them certifies its list as we do. In contrast, we use the existing draft without training, score the whole draft in one pass, and either certify the resulting output list or correct it with the fallback.

Search under budgets. Our certificate (Theorem 1) has the form of the stopping rule of the threshold algorithm for top-K aggregation (Fagin et al., 2003), i.e., the K-th best score seen against a bound on everything unseen. Moreover, the request-dependent cost set by the hidden probability (Section 2) is of the kind that Fagin's instance optimality describes. Similarly, any decoder that certifies its list must score every prefix whose probability exceeds the K-th best item's score, which parallels the nodes that best-first search must expand under any admissible heuristic (Dechter & Pearl, 1985). The additive depth term in the budgeted round bound has the shape of the lower bound for exploring a tree whose structure is not known in advance (Fraigniaud et al., 2006; Dereniowski et al., 2015), which no exploration rule can remove. The catalog trie removes that uncertainty, because the decoder knows in advance which prefixes exist (Papadimitriou & Yannakakis, 1991). Hence, Theorem 2 quantifies how much knowing the trie helps. Finally, the factor that we announce is the same kind of guarantee as in anytime and contract search (Likhachev et al., 2003; Aine et al., 2010), which return a solution within a stated factor of the best one; our plan certifies that factor rather than estimating it. Sequoia (Chen et al., 2024) also chooses its speculation tree under a token budget, but by expected acceptance, whereas our plans are worst-case.

Serving. Our layout relies on paged KV caching (Kwon et al., 2023), prefix sharing (Zheng et al., 2024b), continuous batching (Yu et al., 2022), and fused and block-sparse attention kernels (Dao et al., 2022; Dong et al., 2025), which are orthogonal to the decoder and apply to SIDECAR as they do to beam search.

Impact Statement

This paper presents a decoding procedure for generative recommenders that returns the same list as the model's own ranking at a lower cost in sequential model calls, together with a certificate that states whether the list is exact, or how close to exact it is. It does not change what is recommended, but only how fast and how reliably the model's ranking is produced. Lower decoding cost makes it cheaper to serve such recommenders, and a certificate makes their behavior under a budget auditable, which we view as positive. The societal consequences of recommendation systems themselves, including their effects on exposure and diversity, are the responsibility of the model and of the stack that deploys it, and are not altered by the decoder.

References

Aine, S., Chakrabarti, P. P., and Kumar, R. Contract search: Heuristic search under node expansion constraints. In *ECAI 2010 – 19th European Conference on Artificial Intelligence*, Frontiers in Artificial Intelligence and Applications, pp. 733–738. IOS Press, 2010. doi: 10.3233/978-1-60750-606-5-733.

Ankner, Z., Parthasarathy, R., Nrusimha, A., Rinard, C., Ragan-Kelley, J., and Brandon, W. Hydra: Sequentially-dependent draft heads for Medusa decoding. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=FbhjirzvJG.

Broder, A. Z., Carmel, D., Herscovici, M., Soffer, A., and Zien, J. Efficient query evaluation using a two-level retrieval process. In *Proceedings of the Twelfth International Conference on Information and Knowledge Management*, pp. 426–434. ACM, 2003. doi: 10.1145/956863.956944.

Cai, T., Li, Y., Geng, Z., Peng, H., Lee, J. D., Chen, D., and Dao, T. Medusa: Simple LLM inference acceleration framework with multiple decoding heads. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 5209–5235. PMLR,

2024. URL https://proceedings.mlr.press/v235/cai24b.html.

Chen, C., Borgeaud, S., Irving, G., Lespiau, J.-B., Sifre, L., and Jumper, J. Accelerating large language model decoding with speculative sampling. *arXiv preprint arXiv:2302.01318*, 2023.

Chen, Z., May, A., Svirschevski, R., Huang, Y., Ryabinin, M., Jia, Z., and Chen, B. Sequoia: Scalable and robust speculative decoding. In *Advances in Neural Information Processing Systems*, volume 37, pp. 129531–129563, 2024. doi: 10.52202/079017-4116.

D'Amico, E., De Nadai, M., Chandar, P., Vohra, D., Lin, S., Lefarov, M., Gigioli, P., Penha, G., Kopysitsky, I., Senese, I. J., Mei, D., Fabbri, F., Semerci, O., Zhao, Y., Tang, V., St. Thomas, B., Ranieri, A., Smith, M. N. K., Bernkopf, A., Leung, B., Fazelnia, G., VanMiddlesworth, M., Heath, T. C., Skiden, P. P., Wang, A. Y., Cole, D. J., Damianou, A., Hristakeva, M., Wilbur, R., Chillara, T., Radosavljevic, V., Chitkara, P., Adapa, S., Elenter, J., Huber, B., Wood, J., Vedantam, S., Stypka, J., Ghael, S., Gould, M. D., Murgatroyd, D., Raimond, Y., Lalmas, M., and Bennett, P. N. From habits to discovery: Deploying LLMs for personalized generative recommendations at Spotify. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 7141–7150. ACM, 2026. doi: 10.1145/3770855.3818419.

Dao, T., Fu, D. Y., Ermon, S., Rudra, A., and Ré, C. FlashAttention: Fast and memory-efficient exact attention with IO-awareness. In *Advances in Neural Information Processing Systems*, volume 35, pp. 16344–16359. Curran Associates, Inc., 2022.

De Cao, N., Izacard, G., Riedel, S., and Petroni, F. Autoregressive entity retrieval. In *9th International Conference on Learning Representations (ICLR 2021)*, 2021.

Dechter, R. and Pearl, J. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM*, 32(3): 505–536, 1985. doi: 10.1145/3828.3830.

Deng, J., Wang, S., Cai, K., Ren, L., Hu, Q., Ding, W., Luo, Q., and Zhou, G. OneRec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965*, 2025.

Dereniowski, D., Disser, Y., Kosowski, A., Pająk, D., and Uznański, P. Fast collaborative graph exploration. *Information and Computation*, 243:37–49, 2015. doi: 10.1016/j.ic.2014.12.005.

Ding, S. and Suel, T. Faster top-k document retrieval using block-max indexes. In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 993–1002. ACM, 2011. doi: 10.1145/2009916.2010048.

Ding, Y., Li, J., McAuley, J., and Hou, Y. Inductive generative recommendation via retrieval-based speculation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 40, pp. 14675–14683, 2026. doi: 10.1609/aaai.v40i17.38486.

Dong, J., Feng, B., Guessous, D., Liang, Y., and He, H. Flex Attention: A programming model for generating optimized attention kernels. In *Proceedings of Machine Learning and Systems*, volume 7, 2025. URL https://arxiv.org/abs/2412.05496.

Fagin, R., Lotem, A., and Naor, M. Optimal aggregation algorithms for middleware. *Journal of Computer and System Sciences*, 66(4):614–656, 2003. doi: 10.1016/S0022-0000(03)00026-6.

Fang, D., Gao, J., Li, Y., Zhao, X., and Chang, Y. PRISM: Purified representation and integrated semantic modeling for generative sequential recommendation. *arXiv preprint arXiv:2601.16556*, 2026.

Fraigniaud, P., Gąsieniec, L., Kowalski, D. R., and Pelc, A. Collective tree exploration. *Networks*, 48(3):166–177, 2006. doi: 10.1002/net.20127.

Fu, Y., Bailis, P., Stoica, I., and Zhang, H. Break the sequential dependency of LLM inference using lookahead decoding. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 14060–14079. PMLR, 2024. URL https://proceedings.mlr.press/v235/fu24a.html.

Gao, W., Fan, X., Wang, C., Sun, J., Jia, K., Xiao, W., Ding, R., Bin, X., Yang, H., and Liu, X. Learning an end-to-end structure for retrieval in large-scale recommendations. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, pp. 524–533. ACM, 2021. doi: 10.1145/3459637.3482362.

Geng, S., Liu, S., Fu, Z., Ge, Y., and Zhang, Y. Recommendation as language processing (RLP): A unified pretrain, personalized prompt & predict paradigm (P5). In *Proceedings of the 16th ACM Conference on Recommender Systems*, pp. 299–315. ACM, 2022. doi: 10.1145/3523227.3546767.

Geng, S., Josifoski, M., Peyrard, M., and West, R. Grammar-constrained decoding for structured NLP tasks without finetuning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 10932–10952, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.674.

Gloeckle, F., Idrissi, B. Y., Rozière, B., Lopez-Paz, D., and Synnaeve, G. Better & faster large language models via multi-token prediction. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 15706–15734. PMLR, 2024. URL https://proceedings.

mlr.press/v235/gloeckle24a.html.

He, R. and McAuley, J. Ups and downs: Modeling the visual evolution of fashion trends with one-class collaborative filtering. In *Proceedings of the 25th International Conference on World Wide Web*, pp. 507–517. International World Wide Web Conferences Steering Committee, 2016. doi: 10.1145/2872427.2883037.

He, R., Heldt, L., Hong, L., Keshavan, R., Mao, S., Mehta, N., Su, Z., Tsai, A., Wang, Y., Wang, S.-C., Yi, X., Baugher, L., Cakici, B., Chi, E., Goodrow, C., Han, N., Ma, H., Rosales, R., Van Soest, A., Tandon, D., Wu, S.-L., Yang, W., and Zheng, Y. PLUM: Adapting pre-trained language models for industrial-scale generative recommendations. In *Proceedings of the ACM Web Conference 2026*, pp. 8093–8104. ACM, 2026. doi: 10.1145/3774904.3792802.

He, Z., Zhong, Z., Cai, T., Lee, J. D., and He, D. REST: Retrieval-based speculative decoding. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 1582–1595, Mexico City, Mexico, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.naacl-long.88.

Hou, Y., Li, J., Shin, A., Jeon, J., Santhanam, A., Shao, W., Hassani, K., Yao, N., and McAuley, J. Generating long semantic IDs in parallel for recommendation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 956–966. ACM, 2025. doi: 10.1145/3711896.3736979.

Hu, Y., Wang, D., Li, N., Li, K., She, J., Guo, C., Cai, K., Liu, B., Liu, Y., and Chen, X. Utility-aware generative recommendation: Utility-enriched semantic codes and utility-guided decoding. *arXiv preprint arXiv:2611.02981*, 2026.

Hua, W., Xu, S., Ge, Y., and Zhang, Y. How to index item IDs for recommendation foundation models. In *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, pp. 195–204. ACM, 2023. doi: 10.1145/3624918.3625339.

Jin, B., Zeng, H., Wang, G., Chen, X., Wei, T., Li, R., Wang, Z., Li, Z., Li, Y., Lu, H., Wang, S., Han, J., and Tang, X. Language models as semantic indexers. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 22244–22259. PMLR, 2024. URL https://proceedings.mlr.press/v235/jin24h.html.

Ju, C. M., Collins, L., Neves, L., Kumar, B., Wang, L. Y., Zhao, T., and Shah, N. Generative recommendation with semantic IDs: A practitioner's handbook. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 6420–6425. ACM, 2025. doi: 10.1145/3746252.3761612.

Kim, S., Mangalam, K., Moon, S., Malik, J., Mahoney, M. W., Gholami, A., and Keutzer, K. Speculative decoding with big little decoder. In *Advances in Neural Information Processing Systems*, volume 36, pp. 39236–39256. Curran Associates, Inc., 2023.

Kwon, W., Li, Z., Zhuang, S., Sheng, Y., Zheng, L., Yu, C. H., Gonzalez, J. E., Zhang, H., and Stoica, I. Efficient memory management for large language model serving with PagedAttention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pp. 611–626. ACM, 2023. doi: 10.1145/3600006.3613165.

Lee, D., Kim, C., Kim, S., Cho, M., and Han, W.-S. Autoregressive image generation using residual quantization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 11513–11522, 2022. doi: 10.1109/CVPR52688.2022.01123.

Leviathan, Y., Kalman, M., and Matias, Y. Fast inference from transformers via speculative decoding. In *Proceedings of the 40th International Conference on Machine Learning*, volume 202 of *Proceedings of Machine Learning Research*, pp. 19274–19286. PMLR, 2023. URL https://proceedings.mlr.press/v202/leviathan23a.html.

Li, Y., Wei, F., Zhang, C., and Zhang, H. EAGLE: Speculative sampling requires rethinking feature uncertainty. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 28935–28948. PMLR, 2024a. URL https://proceedings.mlr.press/v235/li24bt.html.

Li, Y., Yang, N., Wang, L., Wei, F., and Li, W. Learning to rank in generative retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8716–8723, 2024b. doi: 10.1609/aaai.v38i8.28717.

Likhachev, M., Gordon, G. J., and Thrun, S. ARA*: Anytime A* with provable bounds on sub-optimality. In *Advances in Neural Information Processing Systems*, volume 16, pp. 767–774. MIT Press, 2003.

Lin, X., Yang, C., Wang, W., Li, Y., Du, C., Feng, F., Ng, S.-K., and Chua, T.-S. Efficient inference for large language model-based generative recommendation. In *The Thirteenth International Conference on Learning Representations*, 2025.

Liu, E., Zheng, B., Ling, C., Hu, L., Li, H., and Zhao, W. X. Generative recommender with end-to-end learnable item tokenization. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 729–739. ACM, 2025. doi: 10.1145/3726302.3729989.

McAuley, J., Targett, C., Shi, Q., and van den Hengel, A. Image-based recommendations on styles and substitutes. In *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 43–52. ACM, 2015. doi: 10.1145/2766462.2767755.

Meister, C., Cotterell, R., and Vieira, T. If beam search is the answer, what was the question? In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2173–2185. Association for Computational Linguistics, 2020a. doi: 10.18653/v1/2020.emnlp-main.170.

Meister, C., Cotterell, R., and Vieira, T. Best-first beam search. *Transactions of the Association for Computational Linguistics*, 8:795–809, 2020b. doi: 10.1162/tacl_a_00346.

Miao, X., Oliaro, G., Zhang, Z., Cheng, X., Wang, Z., Zhang, Z., Wong, R. Y. Y., Zhu, A., Yang, L., Shi, X., Shi, C., Chen, Z., Arfeen, D., Abhyankar, R., and Jia, Z. SpecInfer: Accelerating large language model serving with tree-based speculative inference and verification. In *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 3*, pp. 932–949. ACM, 2024. doi: 10.1145/3620666.3651335.

Morin, F. and Bengio, Y. Hierarchical probabilistic neural network language model. In *Proceedings of the Tenth International Workshop on Artificial Intelligence and Statistics*, volume R5 of *Proceedings of Machine Learning Research*, pp. 246–252. PMLR, 2005. URL https://proceedings.mlr.press/r5/morin05a.html.

Nguyen, T. and Yates, A. Generative retrieval as dense retrieval. In *Gen-IR@SIGIR 2023: The First Workshop on Generative Information Retrieval*, 2023. arXiv:2306.11397.

Oh, S., Amat Gil, F., Smothers, E., Iter, D., Thornburg, M., Argaw, D. M., Medapati, S., Wang, T., Verma, R., Xu, Q., Syal, H., Gholami, E., Bhattacharya, M., and Rastogi, A. Personalized LLM recommendation with semantic IDs: Autoregressive beam search vs. multi-token prediction. *arXiv preprint arXiv:2610.15864*, 2026. Accepted to the NeurIPS 2026 PALM Workshop.

Papadimitriou, C. H. and Yannakakis, M. Shortest paths without a map. *Theoretical Computer Science*, 84(1): 127–150, 1991. doi: 10.1016/0304-3975(91)90263-2.

Pradeep, R., Hui, K., Gupta, J., Lelkes, A., Zhuang, H., Lin, J., Metzler, D., and Tran, V. How does generative retrieval scale to millions of passages? In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1305–1321, Singapore, 2023. Association for Computational Linguistics. doi: 10.18653/v1/2023.emnlp-main.83.

Qu, H., Fan, W., Zhao, Z., and Li, Q. TokenRec: Learning to tokenize ID for LLM-based generative recommendations. *IEEE Transactions on Knowledge and Data Engineering*, 37(10):6216–6231, 2025. doi: 10.1109/TKDE.2025.3599265.

Rajput, S., Mehta, N., Singh, A., Hulikal Keshavan, R., Vu, T., Heldt, L., Hong, L., Tay, Y., Tran, V. Q., Samost, J., Kula, M., Chi, E. H., and Sathiamoorthy, M. Recommender systems with generative retrieval. In *Advances in Neural Information Processing Systems*, volume 36, pp. 10299–10315. Curran Associates, Inc., 2023.

Scholak, T., Schucher, N., and Bahdanau, D. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 9895–9901, Online and Punta Cana, Dominican Republic, 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.779.

Si, Z., Sun, Z., Chen, J., Chen, G., Zang, X., Zheng, K., Song, Y., Zhang, X., Xu, J., and Gai, K. Generative retrieval with semantic tree-structured identifiers and contrastive learning. In *Proceedings of the 2024 Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, pp. 154–163. ACM, 2024. doi: 10.1145/3673791.3698408.

Singh, A., Vu, T., Mehta, N., Keshavan, R., Sathiamoorthy, M., Zheng, Y., Hong, L., Heldt, L., Wei, L., Tandon, D., Chi, E., and Yi, X. Better generalization with semantic IDs: A case study in ranking for recommendations. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pp. 1039–1044. ACM, 2024. doi: 10.1145/3640457.3688190.

Spector, B. and Ré, C. Accelerating LLM inference with staged speculative decoding. In *Workshop on Efficient Systems for Foundation Models (ES-FOMO) at the 40th International Conference on Machine Learning*, 2023. arXiv:2308.04623.

Stahlberg, F. and Byrne, B. On NMT search errors and model errors: Cat got your tongue? In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3356–3362, Hong Kong, China, 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-1331.

Sun, J., Yang, J., Guo, Z., Xu, D., Yan, J., Deng, L., and Zeng, X. UniSGR: Unified framework for semantic ID generation and ranking. *arXiv preprint arXiv:2607.04068*, 2026.

Sun, Z., Suresh, A. T., Ro, J. H., Beirami, A., Jain, H.,

and Yu, F. SpecTr: Fast speculative decoding via optimal transport. In *Advances in Neural Information Processing Systems*, volume 36, pp. 30222–30242. Curran Associates, Inc., 2023. doi: 10.52202/075280-1314.

Tay, Y., Tran, V. Q., Dehghani, M., Ni, J., Bahri, D., Mehta, H., Qin, Z., Hui, K., Zhao, Z., Gupta, J., Schuster, T., Cohen, W. W., and Metzler, D. Transformer memory as a differentiable search index. In *Advances in Neural Information Processing Systems*, volume 35, pp. 21831–21843. Curran Associates, Inc., 2022.

Wang, W., Bao, H., Lin, X., Zhang, J., Li, Y., Feng, F., Ng, S.-K., and Chua, T.-S. Learnable item tokenization for generative recommendation. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pp. 2400–2409. ACM, 2024a. doi: 10.1145/3627673.3679569.

Wang, Y., Hou, Y., Wang, H., Miao, Z., Wu, S., Sun, H., Chen, Q., Xia, Y., Chi, C., Zhao, G., Liu, Z., Xie, X., Sun, H. A., Deng, W., Zhang, Q., and Yang, M. A neural corpus indexer for document retrieval. In *Advances in Neural Information Processing Systems*, volume 35, pp. 25600–25614. Curran Associates, Inc., 2022.

Wang, Y., Xun, J., Hong, M., Zhu, J., Jin, T., Lin, W., Li, H., Li, L., Xia, Y., Zhao, Z., and Dong, Z. EAGER: Two-stream generative recommender with behavior-semantic collaboration. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 3245–3254. ACM, 2024b. doi: 10.1145/3637528.3671775.

Wang, Y., Liu, Z., Tang, G., Yang, Y., Su, S., Liu, J., Zhou, N., Zhou, X., Jiang, W., Liang, J., Lv, X., Tang, R., Hong, L., and Ou, W. Multi-decoder OneRec: Controllable generative retrieval for multi-objective industrial recommendation. *arXiv preprint arXiv:2607.26500*, 2026a.

Wang, Y., Zhou, S., Lu, J., Liu, Z., Liu, L., Wang, M., Zhang, W., Li, F., Su, W., Wang, P., Xu, J., and Zhao, X. NEZHA: A zero-sacrifice and hyperspeed decoding architecture for generative recommendations. In *Proceedings of the ACM Web Conference 2026*, pp. 8073–8082. ACM, 2026b. doi: 10.1145/3774904.3792797.

Wu, S., Ren, Z., Xin, X., Yang, J., Zhang, M., Chen, Z., de Rijke, M., and Ren, P. Constrained auto-regressive decoding constrains generative retrieval. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2429–2440. ACM, 2025. doi: 10.1145/3726302.3729934.

Xie, T., Ku, X., Sun, M., Sha, Y., Wang, L., Wang, P., Wang, Y., Wu, W., Liu, Z., Jiang, P., and Ou, W. From a static multi-level small semantic codebook to a dynamic single-level large semantic codebook for generative recommendation. *arXiv preprint arXiv:2608.21012*, 2026.

Xue, B., Liu, D., Wang, L., Sun, M., Wang, P., Zhang, P., Shi, S., Xu, T., Sha, Y., Liu, Z., Kong, B., Wang, B., Yang, H., Xue, J., Wang, J., Wang, S., Hui, S., Ye, W., Lin, X., Li, Y., Chen, Y., Yin, Z., Chen, Q., Wen, S., Wu, W., Li, H., Zhou, G., Li, C., Jiang, P., and Gai, K. Generative recommendation for large-scale advertising. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 8346–8357, 2026.

Yang, L., Paischer, F., Hassani, K., Li, J., Shao, S., Li, Z. G., He, Y., Feng, X., Noorshams, N., Park, S., Long, B., Nowak, R. D., Gao, X., and Eghbalzadeh, H. Unifying generative and dense retrieval for sequential recommendation. *arXiv preprint arXiv:2411.18814*, 2024.

Yang, Y., Ji, Z., Li, Z., Li, Y., Mo, Z., Ding, Y., Chen, K., Zhang, Z., Li, J., Li, S., and Liu, L. Sparse meets dense: Unified generative recommendations with cascaded sparse-dense representations. In *Advances in Neural Information Processing Systems*, volume 38, pp. 104045–104069, 2025. doi: 10.52202/085713-3135.

Yu, G.-I., Jeong, J. S., Kim, G.-W., Kim, S., and Chun, B.-G. Orca: A distributed serving system for Transformer-Based generative models. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*, pp. 521–538, Carlsbad, CA, 2022. USENIX Association. URL https://www.usenix.org/conference/osdi22/presentation/yu.

Zeghidour, N., Luebs, A., Omran, A., Skoglund, J., and Tagliasacchi, M. SoundStream: An end-to-end neural audio codec. *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 30:495–507, 2022. doi: 10.1109/TASLP.2021.3129994.

Zeng, H., Luo, C., Jin, B., Sarwar, S. M., Wei, T., and Zamani, H. Scalable and effective generative information retrieval. In *Proceedings of the ACM Web Conference 2024*, pp. 1441–1452. ACM, 2024. doi: 10.1145/3589334.3645477.

Zhai, J., Liao, L., Liu, X., Wang, Y., Li, R., Cao, X., Gao, L., Gong, Z., Gu, F., He, J., Lu, Y., and Shi, Y. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 58484–58509. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhai24a.html.

Zhan, T., Fu, K., Lv, C., Lv, Z., and Zhang, S. Semantic trimming and auxiliary multi-step prediction for generative recommendation. *arXiv preprint arXiv:2604.05329*, 2026.

Zhang, J., Wang, J., Li, H., Shou, L., Chen, K., Chen, G., and Mehrotra, S. Draft & verify: Lossless large

language model acceleration via self-speculative decoding. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 11263–11282, Bangkok, Thailand, 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.607.

Zheng, B., Hou, Y., Lu, H., Chen, Y., Zhao, W. X., Chen, M., and Wen, J.-R. Adapting large language models by integrating collaborative semantics for recommendation. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pp. 1435–1448. IEEE, 2024a. doi: 10.1109/ICDE60146.2024.00118.

Zheng, L., Yin, L., Xie, Z., Sun, C., Huang, J., Yu, C. H., Cao, S., Kozyrakis, C., Stoica, I., Gonzalez, J. E., Barrett, C., and Sheng, Y. SGLang: Efficient execution of structured language model programs. In *Advances in Neural Information Processing Systems*, volume 37, pp. 62557–62583. Curran Associates, Inc., 2024b. doi: 10.52202/079017-2000.

Zhou, G., Deng, J., Zhang, J., Cai, K., Ren, L., Luo, Q., Wang, Q., Hu, Q., Huang, R., Wang, S., Ding, W., Li, W., Luo, X., Wang, X., Cheng, Z., Zhang, Z., Zhang, B., Wang, B., Ma, C., Song, C., Wang, C., Wang, D., Meng, D., Yang, F., Zhang, F., Jiang, F., Zhang, F., Wang, G., Zhang, G., Li, H., Hu, H., Lin, H., Cheng, H., Cao, H., Wang, H., Huang, J., Chen, J., Liu, J., Jia, J., Gai, K., Hu, L., Zeng, L., Yu, L., Wang, Q., Zhou, Q., Wang, S., He, S., Yang, S., Yang, S., Huang, S., Wu, T., He, T., Gao, T., Yuan, W., Liang, X., Xu, X., Liu, X., Wang, Y., Wang, Y., Liu, Y., Song, Y., Zhang, Y., Wu, Y., Zhao, Y., and Liu, Z. OneRec technical report. *arXiv preprint arXiv:2506.13695*, 2025.

Zhu, H., Li, X., Zhang, P., Li, G., He, J., Li, H., and Gai, K. Learning tree-based deep model for recommender systems. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pp. 1079–1088. ACM, 2018. doi: 10.1145/3219819.3219826.

Zhu, H., Chang, D., Xu, Z., Zhang, P., Li, X., He, J., Li, H., Xu, J., and Gai, K. Joint optimization of tree-based index and deep model for recommender systems. In *Advances in Neural Information Processing Systems*, volume 32. Curran Associates, Inc., 2019.

Zhu, J., Jin, M., Liu, Q., Qiu, Z., Dong, Z., and Li, X. CoST: Contrastive quantization based semantic tokenization for generative recommendation. In *Proceedings of the 18th ACM Conference on Recommender Systems*, pp. 969–974. ACM, 2024. doi: 10.1145/3640457.3688178.

Zhuang, S., Ren, H., Shou, L., Pei, J., Gong, M., Zuccon, G., and Jiang, D. Bridging the gap between indexing and retrieval for differentiable search index with query generation. *arXiv preprint arXiv:2206.10128*, 2022.

Zhuo, J., Xu, Z., Dai, W., Zhu, H., Li, H., Xu, J., and Gai, K. Learning optimal tree models under beam search. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pp. 11650–11659. PMLR, 2020. URL https://proceedings.mlr.press/v119/zhuo20a.html.

A. Assumptions and proofs

In this appendix, we state the assumptions under which the results of the paper hold, and we prove the results in the order in which they appear in the main text. Each assumption is introduced in the main text where it is first needed.

Assumptions. Throughout the appendix, we assume the following.

- **A1** Every SID has exactly L codes, and the catalog trie is prefix-closed.
- **A2** The scores are constrained to the admissible children of a prefix and sum to one over them; hence, the probability of a prefix equals the total probability of the items below it.
- **A3** The output of the tree pass at a node equals the output of a separate pass over the path of that node.
- **A4** There are no ties among the probabilities that the statements compare.
- **A5** At least K items are scored after the first pass, so that $s_K(0) > 0$.
- **A6** Unless a budget is stated, a fallback round expands every open prefix.

The frontier. Every proof below uses the following property of the frontier, which we state and prove first.

Lemma 1 (frontier)**.** *After any round, every item either has an exact score or lies below exactly one frontier prefix, and in the second case its probability is at most that of the frontier prefix.*

Proof. Fix an item e and follow its SID from the root, which counts as expanded, down to the first prefix of e that is not expanded. Two cases arise.

(a) If every prefix of e up to depth $L-1$ is expanded, then the depth-$(L-1)$ prefix of e is expanded, and hence e has an exact score.

(b) Otherwise, the first prefix of e that is not expanded has an expanded parent, so it is a frontier prefix. By A2, the probability of e is at most the probability of this prefix.

Moreover, no frontier prefix is an ancestor of another, since a frontier prefix is not expanded, whereas the parent of every frontier prefix is. Hence, the frontier prefixes cover disjoint sets of items, and the frontier prefix found in case (b) is unique. □

Proof of Theorem 1. Let e be an item outside E_r. By Lemma 1, e lies below some frontier prefix b, and therefore

$$p(e) \;\le\; p(b) \;\le\; U_r \;\le\; s_K(r),$$

where the first inequality follows from Lemma 1, the second from the definition of U_r as the largest frontier probability, and the third from the assumption of the theorem. Hence, every item outside E_r has a probability of at most the K-th best score in E_r, and since A4 excludes ties, it ranks below the K best items of E_r. Thus, the top-K of E_r is the top-K of the catalog, i.e., SIDECAR-K$_r$ = MODEL-K. □

Proof of Proposition 1. The proposition makes three claims, which we prove in turn.

(a) *s_K never decreases.* A round only adds scored items, so $E_{r+1} \supseteq E_r$, and the K-th best score in a larger set cannot be smaller.

(b) *U never increases.* A round removes every expanded prefix b from the frontier and adds the children of b. By A2, the probabilities of these children sum to $p(b)$, so each of them is at most $p(b)$. Hence, the largest frontier probability cannot rise.

(c) *At most L passes.* First, a frontier prefix that is closed after some round stays closed in every later round, since its probability does not change and, by (a), s_K never decreases. Second, by A6, round $r+1$ expands every prefix that is open after round r. Hence, every prefix that is open after round $r+1$ is a child of a prefix that was open after round r, and the smallest depth among the open prefixes grows by at least one per round. Since frontier prefixes have depths between 1 and $L-1$, no prefix is open after $L-1$ fallback rounds, i.e., after at most L passes. At that point $U \le s_K$, and Theorem 1 applies.

□

Proof of Proposition 2. We prove the three inequalities of the proposition from left to right.

(a) $s_k(r) \le p^*_{(k)}$. Since E_r is a subset of the catalog, its k-th best score cannot exceed the k-th best probability in the catalog.

(b) $p^*_{(k)} \le \max(s_k(r), U_r)$. We distinguish two cases. If $p^*_{(k)} > U_r$, then each of the k best items of the catalog has a probability above U_r. By Lemma 1, an item without an exact score has a probability of at most U_r, so all k of these items are scored, and together with (a) we obtain $s_k(r) = p^*_{(k)}$. Otherwise, $p^*_{(k)} \le U_r$ holds directly. In both cases, $p^*_{(k)} \le \max(s_k(r), U_r)$.

(c) $\max(s_k(r), U_r) \le (1 + \max(\mathrm{gap}_r, 0))\, s_k(r)$. The first term satisfies $s_k(r) \le (1 + \max(\mathrm{gap}_r, 0))\, s_k(r)$ trivially. For the second term, the definition of the gap gives $U_r = (1 + \mathrm{gap}_r)\, s_K(r)$, and since $k \le K$ implies $s_k(r) \ge s_K(r)$, we obtain $U_r \le (1 + \max(\mathrm{gap}_r, 0))\, s_k(r)$.

□

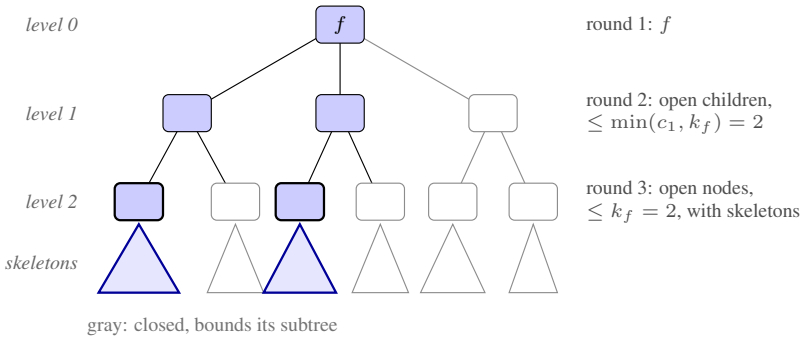


Figure 15. What a plan can cost (Proposition 3), for an open prefix f with $k_f = \lfloor p(f)/t \rfloor = 2$ and the plan $\ell = 2$. At most k_f nodes per level can be above the threshold, and the open nodes (blue) form a top part of the tree, since an open node has an open parent. A closed node (gray) bounds its whole subtree, so nothing below it is fed. The plan feeds the top part one level per round down to level $\ell - 1$, at most $\min(c_j, k_f)$ nodes at level j, and then the open nodes of level ℓ with their whole skeletons (shaded triangles), which gives the two terms of Eq. (2).

Proof of Proposition 3. Figure 15 illustrates both parts of the proof.

(i) Two bounds hold for the nodes at level j below f that are above t and have to be fed.

(a) *The probability bound.* The prefixes at one level below f are disjoint and, by A2, have total probability $p(f)$. Since each prefix above t contributes more than t to this total, at most $\lfloor p(f)/t \rfloor = k_f$ of them exceed t.

(b) *The skeleton bound.* Every node outside the skeleton of f lies on a chain, i.e., its subtree holds a single item, so by A2 its probability equals the probability of that item. The output at the parent of the top node of the chain already gives this probability, so such a node needs no token, whether or not it is above t. Hence, the nodes that have to be fed at level j are among the $c_j(f)$ skeleton nodes at that level.

Together, the two bounds give at most $\min(c_j(f), k_f)$ such nodes at level j.

(ii) Fix any consistent completion (Section 4). Under this completion, the nodes above t form a set that is closed under taking ancestors, since by A2 a node has at least the probability of each of its descendants. The plan ℓ resolves f in two phases.

(a) *Descent.* For the levels $j = 0, \ldots, \ell - 1$, one level per round, the plan feeds the open nodes of level j, of which there are at most $\min(c_j(f), k_f)$ by (i). This gives the first term of Eq. (2).

(b) *Skeleton round.* The plan then feeds the open nodes of level ℓ, of which there are at most k_f, together with their skeletons, which include the nodes themselves. The total size of these skeletons is at most the sum of the k_f largest skeleton sizes at level ℓ, i.e., $\Sigma^{\sigma}_{\ell}(f, t)$, which is the second term of Eq. (2).

After both phases, every node below f has either been fed or lies below a closed node. A closed node bounds every item in its subtree by t, so nothing below f remains open. □

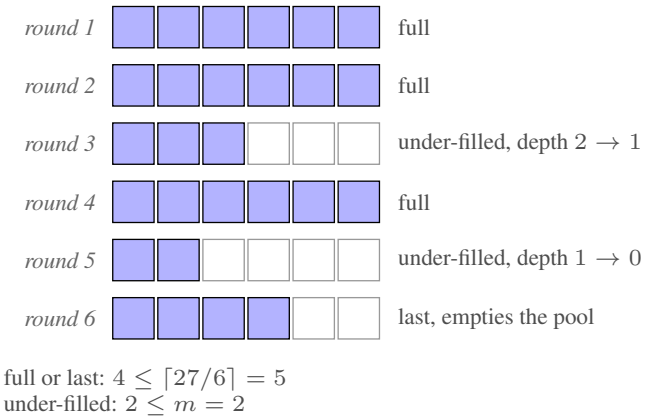


Figure 16. The round bound $\lceil S/M \rceil + m$ (Proposition 4(i)), here with $M = 6$ slots per round, a pool of $S = 27$ nodes and depth $m = 2$. Every round either fills its M slots with pool nodes or runs out of available pool nodes. Rounds of the first kind, together with the last round, number at most $\lceil S/M \rceil$, since every pool node is fed once. A round of the second kind has fed every available pool node, so the remaining depth drops by one, which happens at most m times. The two counts add in whatever order the two kinds occur.

Proof of Proposition 4. (i) We classify every fallback round by the number of pool nodes that it feeds: a round is *full* when it feeds M pool nodes, and *under-filled* otherwise (Fig. 16). We call a pool node *available* once the output at its parent is known, and we bound the two kinds of rounds separately.

(a) *Full rounds.* Every pool node is fed at most once, and the pool holds at most S nodes, so at most $\lceil S/M \rceil$ rounds are full.

(b) *Under-filled rounds.* The procedure never leaves an available pool node unfed while a slot is free, so an under-filled round feeds every available pool node. After such a round, every remaining pool node lies at least one level deeper than before, so the largest number of levels that a remaining plan still descends one round at a time drops by one. Hence, at most m under-filled rounds occur while such levels remain.

(c) *The last round.* Once no such level remains, every remaining pool node is available, so every round is full except possibly the last one, which empties the pool. If this last round feeds $a \geq 1$ nodes, the full rounds number at most $\lfloor (S-a)/M \rfloor \leq \lceil S/M \rceil - 1$, so the last round is already counted in $\lceil S/M \rceil$.

The two counts add up regardless of the order in which full and under-filled rounds occur, which gives the bound $\lceil S/M \rceil + m$.

(ii) Let $B = \lceil S/M \rceil + m$, suppose that $B \leq y$, and let primes denote the quantities after the round. The proof has three steps.

(a) *The schedule fits into one pass.* Since $\lceil S/M \rceil \leq y - m$, we have $y - 1 - m \geq \lceil S/M \rceil - 1$, and therefore

$$x_r \;=\; S - (y-1-m)\,M \;\leq\; S - \big(\lceil S/M \rceil - 1\big)\,M \;\leq\; M.$$

(b) *The bound holds with one round fewer.* Let x denote the number of pool nodes that the round feeds, and assume for now that the pool does not grow and that no plan becomes deeper, i.e., $S' \leq S - x$ and $m' \leq m$; step (c) justifies this. We distinguish two cases.

- If $x \geq x_r$, then $S' \leq S - x_r = (y-1-m)\,M$, so $\lceil S'/M \rceil \leq y - 1 - m$, and together with $m' \leq m$ we obtain $B' \leq y - 1$.
- If $x < x_r$, the round has fed every available pool node. Every remaining node above t is then a child of a node that was fed without its skeleton, i.e., of a node whose plan descends at least one more level. Hence, $m' \leq m - 1$ while $S' \leq S$, and again $B' \leq y - 1$. If $m = 0$, every pool node is available from the start, so in this case the round has fed the whole pool, and the pool is empty afterwards.

(c) *Re-planning does not enlarge the pool.* After the round, the pool and the plans are recomputed, and three facts ensure that this does not increase S or m.

- The threshold can only rise between rounds, since s_K never decreases, and a higher threshold only shrinks the set of nodes above it, and hence the pool.
- For every fed node g, the allotment of the children of g never exceeds the allotment of g minus the node fed. Indeed, their counts k_c satisfy $\sum_c k_c \leq k_g$, because their probabilities sum to $p(g)$, a sum of floors is at most the floor of the sum, and the threshold has not fallen. Moreover, the inequality $\min(a,b) + \min(a',b') \leq \min(a+a', b+b')$ bounds their level counts by those of the parent, and their Σ^σ terms select at most k_g nodes at one level below g.

- Re-choosing the plans after the round can only lower B', since the previous plans remain available.

□

Proof of Theorem 2. (i) Let Δ denote the announced factor, and let $t = (1+\Delta)\, s_K(r)$ denote the corresponding threshold after round r.

(a) By the definition of δ^* in Eq. (6), after the first pass some plans satisfy $\lceil S/M \rceil + m \le y$, where $y = Y - 1$ is the number of remaining rounds.

(b) Every round of BUDGET-SIDECAR feeds at least x_r pool nodes or every available one (lines 6 and 7 of Algorithm 2). Hence, by Proposition 4(ii), the bound holds again with one round fewer after every round, i.e., it is an invariant of the planned run.

(c) When no round remains, the bound reads $\lceil S/M \rceil + m \le 0$, which forces $S = 0$, i.e., no frontier node is above t. Hence, after the last round R we have $U_R \le (1+\Delta)\, s_K(R)$, and Proposition 2 gives (i).

(ii) With $Y = Y_{\min}(M)$ passes, the announced factor is $\delta^*(M, Y) = 0$, so by (i) with $\Delta = 0$, no frontier node is open after $Y_{\min}(M) - 1$ rounds. Hence, Theorem 1 applies, and SIDECAR-K is certified.

(iii) Apply Proposition 4(ii) with the factor announced after round r. It shows that after the round, some plans satisfy $\lceil S'/M \rceil + m' \le y - 1$ at the new threshold. Since δ^*_{r+1} is the smallest factor for which such plans exist with one round fewer, it is at most δ^*_r. □

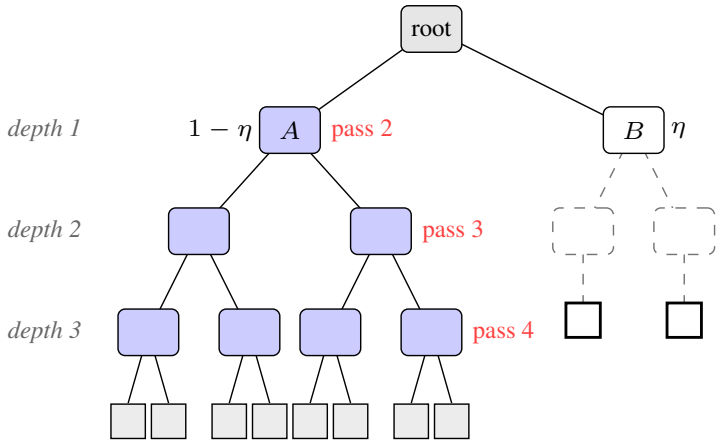


Figure 17. The instance of Proposition 5, with $b = 2$, $L = 4$, $K = 2$ and $M = 7$. The heavy cluster A holds probability $1 - \eta$ spread evenly over its items, and every prefix below it branches, so its skeleton is the whole tree of exactly $\sigma(A) = M$ nodes. The light cluster B holds η on two chains, and its two items are the draft, which the first pass feeds alone, leaving A open. Level-by-level descent (red) reaches the items of A only at pass $L = 4$, so with fewer passes it returns the items of B. BUDGET-SIDECAR finds $R^*(t) = \lceil 7/7 \rceil = 1$ at every threshold, announces exactness after the first pass, and feeds the skeleton of A whole in the second.

Proof of Proposition 5. We construct the instance for any $L \ge 3$, $K \ge 1$, and $b \ge 2$ with $b^{L-1} \ge K$, with $M = (b^{L-1} - 1)/(b - 1)$ nodes per pass and any $2 \le Y \le L - 1$ passes. We then show that the unplanned run, i.e., Algorithm 1 with M nodes per pass, returns after Y passes a list without any item of MODEL-K and is exact only after L passes, whereas BUDGET-SIDECAR announces exactness after the first pass and returns MODEL-K with its certificate after two passes. The proof has four steps.

Step 1: the instance (Fig. 17).

- *The trie.* The root has two children, A and B. Below A, the trie is a complete b-ary tree: every prefix of depth 1 to $L - 1$ below A has b children, and the prefixes of depth L are the items. Hence, A holds b^{L-1} items, every prefix below A is

branching, and the skeleton of A is the whole tree of internal nodes, $\sigma(A) = 1+b+\cdots+b^{L-2} = (b^{L-1}-1)/(b-1) = M$. Below B, there are K children, each of which is a chain down to a single item, so B holds K items.

- *The model.* The model puts probability η on B, split evenly over its K items, and probability $1-\eta$ on A, split over its items with pairwise distinct probabilities that lie within a factor of two of each other. For example, uniform conditionals that are perturbed slightly satisfy A2 and A4. We choose η small enough that every item below A has a probability above η/K. Then MODEL-K consists of K items below A, and since the conditionals below A are nearly uniform, these items lie below distinct depth-$(L-1)$ prefixes.
- *The draft.* The draft consists of the K items below B.

Step 2: the first pass, which is common to both runs. The candidate trie is B with its K chain children, so the first pass with the chain rules feeds B alone, and the chain rules resolve the K items below B exactly. Hence, E_0 is the draft, $s_K(0) = \eta/K$, and the frontier is $F_0 = \{A\}$ with $p(A) = 1-\eta > s_K(0)$, so A is open and the certificate fails.

Step 3: the unplanned run.

- Every prefix below A is branching, so no chain rule applies, and every fallback round feeds the open prefixes one level deeper. After r rounds, the fed prefixes below A are therefore those of depths $1, \ldots, r$, with at most $b^{L-2} \le M$ of them per round, so the budget never binds.
- An item below A is scored only when its depth-$(L-1)$ prefix is fed, which first happens after $L-1$ rounds, i.e., after L passes.
- Hence, after $Y \le L-1$ passes, the scored set is still the draft, so the returned list consists of the K items below B and contains no item of MODEL-K. Moreover, the frontier holds prefixes of probability about $(1-\eta)/b^{Y-1} > s_K$, so the certificate fails as well.
- After L passes, every item below A is scored, the list equals MODEL-K, and the frontier is empty.

Step 4: the planned run (BUDGET-SIDECAR).

- *The announcement.* At any threshold $t = (1+\delta)\, s_K(0) < 1-\eta$, the only open prefix is A, and its plan $\ell = 0$ costs $S_0(A,t) = \sigma(A) = M$ by Eq. (2). Hence, $S_0^*(t) \le M$ and $R^*(t) \le \lceil M/M \rceil + 0 = 1 \le Y-1$ for every $Y \ge 2$. It follows that $\delta^*(M,Y) = 0$ and $Y_{\min}(M) = 2$, i.e., exactness is announced after the first pass.
- *The first fallback round.* The pool is A with its skeleton, so $S = M$ and $m = 0$, and the schedule demands $x_1 = S - (Y-2)\, M \le M$ pool nodes. The batch therefore holds A followed by its $M-1$ remaining skeleton nodes, which fit into the pass.
- *The result.* After this round, every depth-$(L-1)$ prefix below A has been fed, every item below A is scored, and no frontier prefix remains. Hence, $U_1 = 0$, the certificate holds, and SIDECAR-K$_1$ = MODEL-K. Thus, the planned run returns MODEL-K with its certificate after two passes, for every $Y \ge 2$.

□

B. Further guarantees

Building upon the results of Sections 2 to 4, this appendix states further guarantees that sharpen them. The main text does not use these guarantees.

Alignment by depth. Let μ_j be the probability of the items whose first j codes match no candidate of the draft, so that $\mu_0 = 0 \le \mu_1 \le \cdots \le \mu_{L-1} = \varepsilon$. These quantities measure how the draft is aligned with the model, depth by depth, and they refine the hidden-probability condition of Section 2.

Proposition 6 (alignment by depth)**.** (i) *The certificate holds after one pass if $U_0 < p^*_{(K)}$, and only if $U_0 \le p^*_{(K)}$.*
(ii) *After r fallback rounds,* SIDECAR-K$_r$ *misses at most $\lfloor \mu_{L-1-r}/p^*_{(K)} \rfloor$ items of* MODEL-K.
(iii) *Every fallback round expands at most $\lfloor \varepsilon/s_K(0) \rfloor$ prefixes.*

Part (i) is the sharp form of that condition: the first pass certifies exactly when the heaviest uncovered prefix is lighter than the K-th best item, and Fig. 4 shows the condition on the benchmark. Part (ii) says that each fallback round repairs one level of misalignment. An item that the draft missed at depth j is reached after $L-j$ rounds, and the probability left uncovered at that depth bounds how many such items can exist. Part (iii) bounds the number of tokens that a round feeds.

Proof. (iii) Every open prefix after round r has probability above $s_K(r) \geq s_K(0)$. Open prefixes are disjoint, since they are frontier prefixes, and their total probability is at most ε, because every item below a frontier prefix is unscored after the first pass and stays unscored below an unexpanded prefix. Hence, fewer than $\varepsilon/s_K(0)$ of them exist.

(ii) Let e be an item of MODEL-K that is missing from SIDECAR-K$_r$. Then e is unscored, so it lies below a frontier prefix b of the first pass, of some depth d, and $p(b) \geq p(e) \geq p^*_{(K)} \geq s_K(r')$ for every round r', with strict inequality while e is unscored by A4. So b is open after the first pass and round one expands it (A6). The depth-$(d+1)$ prefix of e is then open and round two expands it, and so on, until the depth-$(L-1)$ prefix of e is expanded and e is scored. This takes $L-d$ rounds. Since e is still missing after r rounds, $d \leq L-1-r$, which means that the first $L-1-r$ codes of e match no candidate. The missing items are therefore distinct items of probability at least $p^*_{(K)}$ among a set of total probability μ_{L-1-r} (A2), so at most $\lfloor \mu_{L-1-r}/p^*_{(K)} \rfloor$ of them exist.

(i) If $U_0 < p^*_{(K)}$, every item of the exact top-K is scored, since an unscored one would lie below a frontier prefix of probability at least $p^*_{(K)}$. Hence, $s_K(0) = p^*_{(K)} > U_0$ and the certificate holds. Conversely, the certificate gives $U_0 \leq s_K(0) \leq p^*_{(K)}$. □

Exact top ranks of an early-stopped list. Proposition 2 bounds every rank of a list returned before the certificate holds. In fact, the top ranks of such a list are already exact.

Proposition 7 (exact top ranks)**.** *After any round r, if $s_k(r) \geq U_r$ for some $k \leq K$, the first k returned items are exactly the model's top-k.*

SIDECAR-K$_r$ therefore agrees with MODEL-K down to the first rank whose score falls below U_r, and only the ranks below it are bounded by the factor of Proposition 2 rather than exact.

Proof. If $s_k(r) \geq U_r$, every unscored item has probability at most $s_k(r)$, so the k best scored items are the k best of the catalog (A4). □

Announcement without the trie. It helps to see what the catalog trie contributes to the announcement of Section 4. Without it, the only information about the nodes below an open frontier prefix f is the probability bound of Proposition 3(i), at most $k_f = \lfloor p(f)/t \rfloor$ open nodes at any level, and the number n_f of items below f, which caps the nodes at any level as well. The fallback can then only go level by level, and the open prefixes must fit in a single round at every level. This gives the count

$$\widehat{N}(t) = \sum_{f \in F_0 \colon p(f) > t} \Big(1 + (L - 1 - \mathrm{depth}(f))\, \min\big(n_f, k_f\big)\Big) \tag{7}$$

of prefixes that can ever be open, and hence the announcement $\delta^*(M) = \min\{\delta : \widehat{N}((1+\delta)\, s_K(0)) < M\}$ for $Y = L$ passes. The trie improves on this in two places: $c_j(f)$ replaces n_f as the structural cap, and the skeleton plans add a second way to resolve an open prefix within the budget, namely by feeding its whole skeleton in one round. The following proposition shows that the trie never loosens the announcement.

Proposition 8 (the announcement without the trie)**.** (i) *With M nodes per pass and $Y = L$ passes, the level-by-level fallback returns a list within a factor $1 + \delta^*(M)$ of the exact top-K at every rank.*
(ii) *$\delta^*(M, L) \leq \delta^*(M)$: the announcement with the trie is never looser than the one without it.*

Figure 11(c) in Section 5.7 compares the two announcements on the benchmark.

Proof. (i) Fix a threshold $t > 0$ and a frontier prefix f of the first pass with $p(f) > t$. At each level below f the prefixes are disjoint with total probability $p(f)$ (A2), so at most $\lfloor p(f)/t \rfloor$ of them exceed t, and at most n_f exist. A prefix that is ever open under tolerance δ has probability above $(1+\delta)\, s_K(r) \geq (1+\delta)\, s_K(0)$ and is either a first-pass frontier prefix or a descendant of one. Hence, at most $\widehat{N}((1+\delta)\, s_K(0))$ prefixes, Eq. (7), are ever open. If this number is below M, the prefixes above the tolerance always fit in a round, they are the heaviest open prefixes, so the round expands all of them, and they descend one level per round as in the proof of Proposition 1. After $L-1$ rounds none is open, so $U_R \leq (1+\delta)\, s_K(R)$ and Proposition 2 gives the factor $1+\delta$. This is the announcement $\delta^*(M)$.

(ii) Take the largest plan for every frontier prefix, pure level-by-level descent. Since $c_0(f) = 1$, every skeleton node has an item below it, so $c_j(f) \leq n_f$, and at most $L - 1 - \mathrm{depth}(f)$ levels lie below f, the plan's cost is at most $\min\big(\sigma(f),\ 1 + (L - 1 - \mathrm{depth}(f)) \min(n_f, k_f)\big)$, which is at most the term of f in Eq. (7). Summing, $S \leq \widehat{N}(t)$, so

$\widehat{N}(t) < M$ gives $\lceil S/M \rceil + m \le 1 + (L-2) = L-1$: the condition of the count without the trie implies ours, hence $\delta^*(M, L) \le \delta^*(M)$. □

Monotonicity of the announcement.

Proposition 9 (monotonicity)**.** (i) *$\delta^*(M, Y)$ is non-increasing in M and in Y.*

(ii) *$\delta^*(M, 2)$ is the smallest δ for which the skeletons of the prefixes open under δ fit in one pass of M nodes, which is the two-pass bound of the skeleton round (Section 3) under a memory budget.*

Part (i) says that more memory or more passes never make the guarantee worse. Part (ii) covers the case $Y = 2$, in which one fallback round remains, so only the plan $\ell = 0$ fits and the pool is the union of the skeletons of the open prefixes.

Proof. (i) The bound $\lceil S/M \rceil + m$ of Proposition 4 is non-increasing in M, and the condition $B \le Y - 1$ relaxes as Y grows, so the smallest feasible δ cannot increase in either.

(ii) With $Y = 2$ one round remains, so $m = 0$ and $\lceil S/M \rceil \le 1$ with $S_0(f, t) = \sigma(f)$. Hence, the skeletons of the prefixes above t must fit in M, and the smallest such δ is the announcement. □

C. Computing the announced factor

Equation (6) defines the announced factor as the smallest δ whose round bound fits the budget. This appendix explains what in that computation depends on the request, what can be stored ahead of time, and how the minimum is found.

Precomputed and per-request quantities. The round bound $R^*(t)$ of Eq. (5) is built from two kinds of quantities. The counts $c_j(f)$ and the skeleton sizes at each level below a prefix f depend only on the catalog trie. They are computed once for every internal node, with the skeleton sizes of each level sorted in decreasing order and stored as prefix sums, so that the sum of the k largest is one array lookup. The frontier F_0, its probabilities $p(f)$ and the score $s_K(0)$ depend on the request and are known after the first pass. The threshold t enters $R^*(t)$ in only two places: it decides which prefixes are open, $p(f) > t$, and it sets the integer $k_f = \lfloor p(f)/t \rfloor$ of each open prefix. For a fixed f, the cost $S_\ell(f, t)$ of Eq. (2) is therefore a function of the integer k_f alone. Moreover, this function stops changing once k_f is large enough. Once k_f reaches $K_f = \max_j c_j(f)$, every term $\min(c_j(f), k_f)$ equals $c_j(f)$ and the sum of the k_f largest skeleton sizes is the sum of the whole level, so a larger k_f changes nothing. One evaluation of $R^*(t)$ is thus a gather of one row per open prefix from the stored tables, followed by a minimum over the caps $m \le L - 2$, and it costs $O(|F_0| \cdot L)$ operations. If desired, the table $T_f[m][k] = \min_{\ell \le m} S_\ell(f, k)$ for $k \le K_f$ can also be stored per node, which reduces the evaluation to lookups; its total size is at most $(L-1) \sum_f \sigma(f)$ entries.

Bisection. $R^*(t)$ is non-increasing in t, so the thresholds that fit the budget form an interval that extends to the right. The implementation bisects on t between $s_K(0)$ and $\max_{f \in F_0} p(f)$, keeps the feasible upper end of the bracket, and stops after a fixed number of iterations, twenty in our runs. The returned threshold is always feasible, so the announcement it yields is valid, and it lies within 2^{-20} of the bracket above the exact infimum.

Exact search over the breakpoints. Because t acts only through the open set and the integers k_f, the function $R^*(t)$ is a step function. It can change only at the thresholds where some k_f changes or some prefix closes, namely at the *breakpoints*

$$B = \{\, p(f)/k : f \in F_0,\ 1 \le k \le K_f \,\} \cup \{\, p(f) : f \in F_0 \,\}, \tag{8}$$

and it is constant between consecutive breakpoints, since $k_f(t) = k$ for $t \in (p(f)/(k+1),\ p(f)/k]$. Breakpoints with $k > K_f$ are omitted because the cost has saturated there. Hence, $|B| \le \sum_{f \in F_0} (K_f + 1)$, which is a few thousand on the benchmark. After sorting B, a binary search finds the first breakpoint whose R^* fits $Y - 1$ in $\lceil \log_2 |B| \rceil$ evaluations, and the result is exact. One detail follows from the strict inequality in the definition of an open prefix: the thresholds that fit form an interval that is open at its lower end, so δ^* in Eq. (6) is an infimum, and the planned run uses the first representable threshold above the breakpoint, which changes the factor by a rounding unit.

All budgets in one sweep. The breakpoints also give the announced factor for every budget in a single sweep. Walking B in increasing order, each step either lowers one k_f by one or closes one prefix, so each $S^*_m(t)$ is updated by a single table difference per cap m, and $R^*(t)$ is recomputed as a minimum over m. One sweep therefore yields $R^*(t)$ at every breakpoint in $O(|B| \cdot L)$ time after the sort. From this curve one reads $\delta^*(M, Y)$ for every Y, the number $Y_{\min}(M)$, and, since only

the ceiling in Eq. (5) depends on M, the same quantities for several memory budgets without recomputation. The same sweep serves the re-planning after each round, applied to the frontier F_r.

Computational cost. On the benchmark, the frontier has hundreds of prefixes and B a few thousand entries, so either method takes microseconds to milliseconds on a CPU, against a model pass of milliseconds on a GPU. The choice between bisection and the exact search is therefore a matter of exactness, not of latency.

D. Details of experiments

Data, SIDs and model. We use Amazon Reviews 2014, category Beauty, with the 5-core filter and the leave-one-out split, which gives 22,363 users, 12,101 items and 198,502 interactions. Item text (title, brand, category, description) is embedded with a sentence encoder, and an RQ-VAE with three codebooks of 256 codes each, trained with $\beta = 0.25$ and k-means initialization, assigns the first three codes, and a collision code separates items with equal content codes. The catalog trie has 256, 9,043 and 11,792 internal nodes at depths 1 to 3. Of these, 78% of the depth-2 and 98% of the depth-3 nodes have a single child, so the skeleton of the catalog has 2,533 nodes. The generator is a four-layer decoder-only transformer ($d = 128$, four heads, learned positions, about a million parameters) trained with the next-token loss on flattened SID histories of up to 20 items. Its Hit@10 under exact decoding is 0.055, which is in the range of published results for this setup.

Evaluation protocol. For every test user, one tree pass over all 21,091 internal nodes of the catalog trie gives the model's exact conditional distribution at every node. Every decoder, draft and metric in the paper is then a deterministic replay on these tables, which is what makes it possible to check every proven statement on every request. We also verified the replay against the real model. Tree passes over ordinary sequences agree to within 10^{-5} in the largest logit difference, and Algorithm 1 on the real model returns MODEL-K on every test user with the number of passes that the replay predicts.

Drafts. The model-free drafts are built from the leave-one-out training parts of the sequences and queried with the history stale by one interaction. Content kNN scores every item by its largest cosine similarity, in the L2-normalized content embedding, to any of the last five history items. Co-occurrence counts, for every ordered pair of items at most five positions apart in a training sequence, how often the second follows the first, row-normalizes the counts, and scores an item by the recency-weighted sum of these rates over the last five history items. Matrix factorization is implicit-feedback weighted ALS with 64 factors, confidence weight 40 and 15 sweeps on the binary user–item matrix, with the user's learned vector scored against every item. All of them exclude the history items themselves and keep the top N. The quantized user sequence is the top-N of a copy of the model whose two-dimensional weights are rounded, per output row, to FP4 (E2M1), run on the same one-interaction-old history; its weights are 11% off in relative Frobenius norm and its top-10 agrees with the full model's on 69% of requests. An FP8 (E4M3) copy, with weights 2.6% off and 92% top-10 agreement, gave a sequence indistinguishable from the full model's in every measure, within 0.01 passes at every size, and is omitted from the figures. Write a user's sequence as $s_1, \ldots, s_{n-1}, s_n$ with s_n the held-out target, so that the request's history, which the model decodes from, is $s_1 \ldots s_{n-1}$. The user sequence is the model's top-N for $s_1 \ldots s_{n-2}$, which is what a batch job that ran before the newest interaction s_{n-1} would have stored, so it is stale by one interaction. The per-user list cannot contain s_{n-1} without a model call for this user after that interaction, which is the online cost the method avoids. Hence, one interaction is the least staleness any per-user batch artifact has. Finally, no draft contains information about the held-out item s_n. The flatter models apply a temperature to every conditional distribution of the trained model, and the stored sequence of a flatter model is recomputed from that model.

Budgets. Under a budget of M nodes per pass the first pass feeds the skeleton members of the candidate trie, added in draft order while they fit. Since most candidates lie on chains below a level-1 cluster, a candidate costs a token only when its cluster or a branching prefix on its path is new. As a result, $M = 16$ holds about 90 candidates of the user sequence, $M = 64$ about 210 and $M = 256$ about 440 of its 500. The unplanned run in the ablations is Algorithm 1 with M nodes per pass, i.e., BUDGET-SIDECAR with the largest plan for every prefix and without line 7 of Algorithm 2, which amounts to level-by-level descent with no schedule and no speculation. Every figure of Section 5 uses the first 10,000 users.

Equal resources with the MF draft. Figure 18 repeats the equal-resource comparison of RQ5 (Fig. 9) for the MF draft. On the trained model, the MF draft is exact a few percent less often than BEAM-K at the smallest memories, because the clusters that its first candidates point to are a worse start than the best level-1 codes of the model. From a moderate memory on, it is exact and certified on nearly every request, like the user sequence. On the flat models, however, the MF draft does not share the benefit that flatness gives the user sequence (Section 5.5). The small-subtree items that a flat model ranks

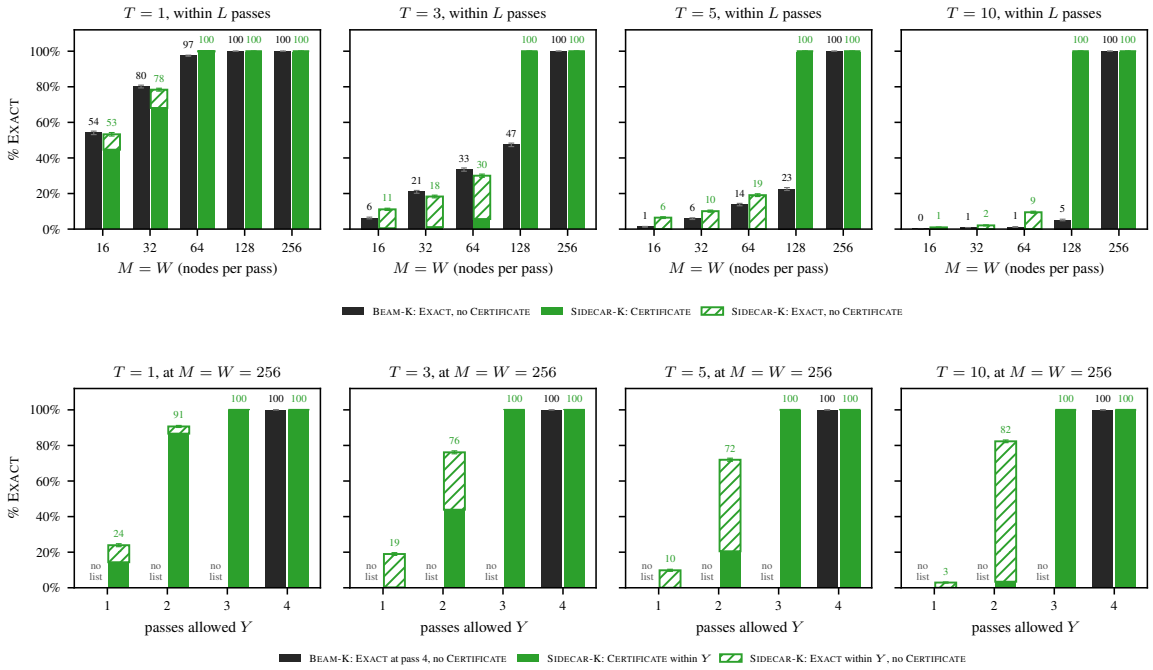


Figure 18. Figure 9 for the MF draft. The MF draft is certified after three passes at beam search's memory, but at small memories it is exact somewhat less often than BEAM-K, on the trained model and at $T = 3$. A draft that ignores the model therefore still reaches a certified list within beam search's budget, but it does not share the advantage that flatness gives the user sequence.

highest are not the items that users with the same history went on to buy, so the MF draft rarely contains them, and the fallback has to find them.

E. Frequently asked questions

1. **Does the draft have to contain the top-K?** No. Section 2 shows that the first pass is certified when the probability that the draft leaves uncovered is below the probability of the K-th best item, and its sharp form, Proposition 6(i), says that the certificate holds exactly when the heaviest uncovered prefix is below it. A draft therefore only has to land in the right prefixes, since the first pass also scores the siblings of the candidates without additional tokens, and the fallback finds the remaining items. This is also why recall against a held-out item is the wrong measure of a draft. Recall counts whether one item is present, whereas the cost depends on how much probability the draft leaves uncovered.
2. **What if the draft has fewer than K items, the size of the list?** Constrained beam search, in TIGER and in the industrial systems of Section 6 alike, needs a width of at least K to return K items, and in practice it runs a width well above K. A draft of at least K items, which a first-stage retriever or a batch job produces without any model call at request time, is therefore a mild assumption. Nevertheless, nothing in the guarantees requires $N \geq K$, and we also test the extreme case $N < K$, in which two properties keep the decoder well behaved. First, the first pass scores more items than the draft holds, since feeding the branching prefixes of one candidate scores every item on a chain below them; on our benchmark, a single candidate already fills the list of ten after the first pass on nearly every request, except for the user sequence on the flattest models. Second, on the requests where the first pass nevertheless scores fewer than K items, assumption A5 of Appendix A is violated; the K-th best score is then zero, so every frontier prefix is open and the fallback first fills the list. The list therefore remains exact within L passes for every draft (Proposition 1), whereas the announced factor is finite only from the round that completes the list on.
   Figure 19 compares the unplanned run with drafts of one to ten items, and of 500 for reference, against beam search at equal resources. On the trained model, the size of the draft barely matters: even a single candidate is about as good as BEAM-K at every memory and exact far more often at the smallest one. On the flatter models, however, a draft of a few items gives up most of the advantage that the stored sequence of 500 entries has at small memory, since a few candidates cannot point to all the clusters that hold the top-10, and at intermediate memories it is exact somewhat less often than BEAM-K. From $M = 128$ on, every draft is exact on nearly every request at every temperature, whereas beam search needs a width of 256. Thus, a deployment should keep N well above K, but a short draft reduces the share of exact lists

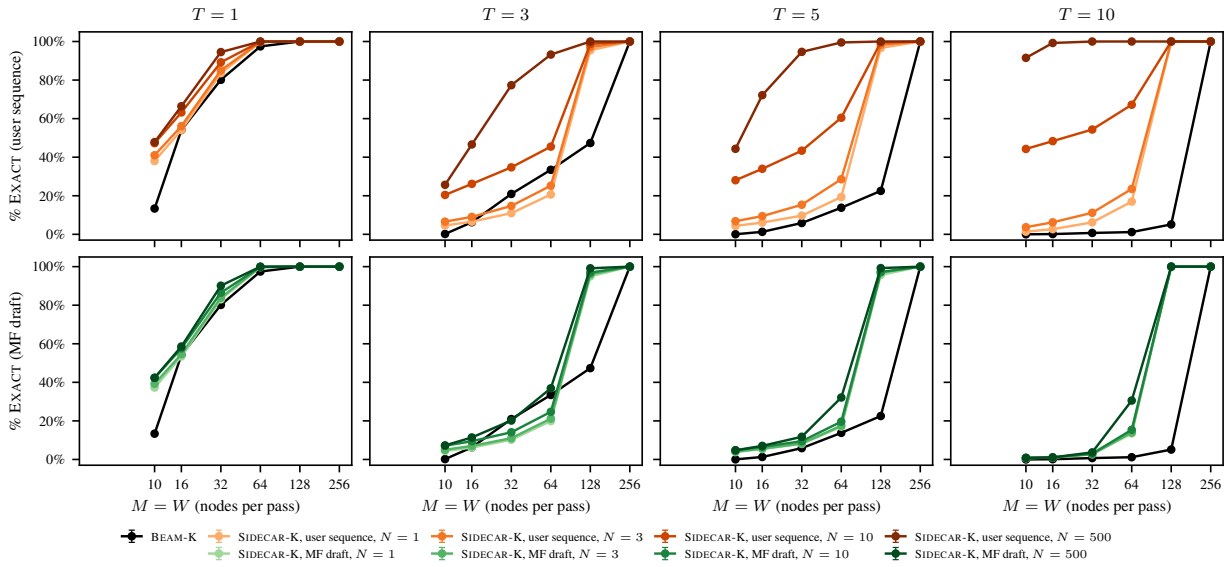


Figure 19. Drafts smaller than the list, at beam search's resources ($K = 10$, $Y = L$). Even a single candidate matches BEAM-K on the trained model, and from $M = 128$ on every draft is exact on nearly every request, whereas beam search needs the full width. Below that memory, a short draft gives up most of the advantage that the stored sequence has on the flatter models.

only at small memory, and it never makes the certificate wrong.

3. **Why does the procedure ever speculate, if speculation costs typical-case accuracy?** Speculation is what makes a guarantee for the worst case possible. Without it, the planned run becomes the unplanned run, i.e., Algorithm 1 with M nodes per pass, and nothing bounds its gap before the last pass. On the instance of Proposition 5, for example, the unplanned run returns no item of MODEL-K, even though the budget suffices for exactness. The reason is that a heavy open prefix whose resolution depth exceeds the remaining rounds cannot be resolved one level at a time. It can only be resolved in time if its skeleton is fed before the model's output at those levels is known. The schedule of Eq. (4) forces exactly as much of this as the worst case needs and no more, and the procedure fills every other slot with prefixes that are known to be open. In exchange, the factor $1 + \delta^*$ holds within the budget against the adversary of Section 4, and the measured price in typical-case overlap is small (Section 5.7).
4. **Is this specific to recommendation?** The method uses three properties: outputs of fixed length over a prefix-closed vocabulary, constrained scores whose prefix probabilities bound the items below, and a draft of plausible outputs. Any generative retriever over document identifiers has the first two, and most have a first-stage retriever that supplies the third. What is specific to SIDs, then, is the small L, which makes the fallback bound of L passes a strong statement, and the unary structure of RQ-VAE tries, which makes the skeleton small.
5. **What would change at production scale?** The catalog trie would have thousands of level-1 clusters and a skeleton far larger than a pass can hold, so the unlimited-memory results no longer apply, and the budgeted results of Section 4 describe the decoder. The draft then has to be good enough to cover the model's probability, which the stored sequence does by construction, and the memory per pass has to cover the open clusters. The announcement states after the first pass whether it does.