AGR: AGENTIC GENERATIVE RETRIEVAL WITH TOOL-AUGMENTED SEARCH

Anonymous authors
Paper under double-blind review

ABSTRACT

Large-scale search and recommendation systems derive much of their retrieval quality from rich, heterogeneous features, yet existing generative retrieval (GR) methods struggle to exploit such feature ecosystems efficiently. Static feature injection either encodes all available signals or places them according to predetermined schemes, increasing serving cost and introducing irrelevant or conflicting evidence. More fundamentally, existing methods cannot adaptively determine when a feature is needed, which feature to select, or where it should influence hierarchical Semantic ID (SID) generation. We propose *Agentic Generative Retrieval (AGR)*, a tool-augmented paradigm that interleaves SID generation with on-demand feature acquisition. Conditioned on the request and decoded SID prefix, *AGR* decides whether to invoke a tool, which feature to acquire, and where its observation should influence subsequent decoding, thereby steering generation toward feature-consistent SID branches. To supervise this behavior, we construct tool-augmented trajectories based on feature applicability and information gain along the SID hierarchy, and train the model through SID pretraining followed by tool-augmented fine-tuning. Experiments on an industrial dataset and three public benchmarks, together with an online A/B test, consistently demonstrate the effectiveness of *AGR*.

1 INTRODUCTION

Generative Retrieval (GR) casts retrieval as autoregressive identifier generation (Tay et al., 2022; Wang et al., 2022; Bevilacqua et al., 2022). Following TIGER (Rajput et al., 2023), items are commonly represented by hierarchical Semantic IDs (SIDs) learned through residual quantization. This paradigm has also been successfully adopted in large-scale industrial scenarios (Deng et al., 2025; Chen et al., 2025a;c; Team et al., 2026).

Deploying GR in industrial systems, however, exposes a fundamental mismatch between request-conditioned SID generation and the feature-intensive nature of such systems (Zhai et al., 2024). Large-scale retrieval depends heavily on heterogeneous signals, such as user profiles, behavioral histories, and freshness cues (Hou et al., 2026). Yet existing GR methods scale poorly with the number of features (Figure 1), typically appending all signals to the input or injecting them according to a predetermined scheme, making it difficult to jointly maintain retrieval quality and serving efficiency (Zhou et al., 2026). Specifically, encoding all features indiscriminately inflates serving cost and introduces irrelevant or conflicting evidence, as feature utility is inherently request-dependent: a feature indispensable for one request may be pure noise for another (Zhang et al., 2026a). This raises the question of **when a feature should be incorporated**. Meanwhile, injecting features at a fixed scheme cannot adapt to the decoding state, but residual quantization induces a hierarchical SID space where different depths resolve distinctions at different granularities. Therefore, a feature highly informative at one level may be irrelevant at others. This raises the question of **where a feature should influence generation**. Addressing both questions is crucial for efficient and scalable GR deployment in industrial search, yet effective feature injection remains largely overlooked.

To this end, we propose **Agentic Generative Retrieval (AGR)**, which reformulates generative retrieval as an interleaved process of SID generation and feature acquisition. *AGR* encapsulates heterogeneous features as tools, each uniquely identified by a tool token. Conditioned on the current decoding state, the model decides whether to emit a tool token and, if so, which tool to invoke. The

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

acquired feature values are then represented as observation tokens that reshape the probability distribution over subsequent SID tokens, steering generation toward feature-consistent paths. As features are introduced only when the decoding state warrants them, *AGR* reduces feature noise and naturally aligns feature granularity with the coarse-to-fine progression of SID generation. Through this adaptive tool use, *AGR* shifts generative retrieval from passive SID generation to agentic decision-making over both items and features.

Contributions. Our contributions are shown as follows:

- Driven by the industrial need to exploit rich, heterogeneous feature ecosystems in GR, we systematically reveal and quantify two fundamental limitations of static feature injection in GR: it is neither request-adaptive (*when* to inject) nor level-adaptive (*where* to inject). We further argue that features should be acquired on demand and injected at the right SID depth.
- We propose *Agentic Generative Retrieval (AGR)*, which encapsulates feature injection as tool invocation and thereby unifies adaptive feature selection with dynamic placement into a single decoding process, paving the way for exploiting rich features in real-world retrieval systems.
- We derive principled criteria for choosing which tool to invoke and where it is most informative along the SID hierarchy, so that each tool call steers decoding toward feature-consistent branches, and instantiate them in a two-stage pipeline of SID pretraining and tool-augmented fine-tuning.
- We evaluate *AGR* on a billion-scale industrial retrieval system and three widely used public benchmarks. *AGR* improves recall by up to +14.04% over baseline and yields consistent online A/B gains. Extensive experimental analyses further explain the gains achieved by *AGR*.

2 Problem Definition

For reference, the notation used throughout this paper is summarized in Table 7.

Generative Retrieval. Given a request containing query q and model-visible user context u_{vis}, generative retrieval aims to retrieve K items from a corpus $\mathcal{X}$ by directly generating item identifiers. Each item $x \in \mathcal{X}$ is represented by a hierarchical SID sequence $\mathbf{s} = [s_1, \ldots, s_L]$, where $s_i \in \mathcal{S}_i$ denotes the token at level i and $\mathbf{s}_{\leqslant i} = [s_1, \ldots, s_i]$ denotes its prefix. The retriever generates the sequence autoregressively by modeling $\mathbb{P}_\theta(s_{i+1} \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i})$, and scores each prefix by accumulating token-level log-probabilities $\pi_\theta(\mathbf{s}_{\leqslant i}) = \sum_{j=1}^{i} \log \mathbb{P}_\theta\left(s_j \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant j-1}\right)$. During inference, a prefix trie Tr over valid SIDs constrains beam search to valid paths only. After L decoding steps, the top-K completed sequences are mapped to the corresponding items as results (Cao et al., 2021).

Feature Injection. Beyond the visible request context q and u_{vis}, practical GR systems often rely on heterogeneous auxiliary features (e.g., long-term behaviors, user attributes, etc), stored in external services. We denote the candidate feature set as $\mathcal{F}$. Incorporating a subset $\mathcal{F}' \subseteq \mathcal{F}$ into decoding is referred to as *feature injection*, which falls into two categories:

- **Static injection** commits to a fixed subset $\mathcal{F}_{\text{static}} \subseteq \mathcal{F}$ before decoding and reuses it at every step, so the same features influence the entire SID generation:

$$\pi_\theta(\mathbf{s}) = \sum_{i=1}^{L} \log \mathbb{P}_\theta(s_i \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i-1}, \mathcal{F}_{\text{static}}). \tag{1}$$

- **Dynamic injection** selects a different subset at each step conditioned on the decoded prefix $\mathbf{s}_{x,<i}$, allowing both the identity and the number of injected features to vary across decoding depths:

$$\pi_\theta(\mathbf{s}) = \sum_{i=1}^{L} \log \mathbb{P}_\theta(s_i \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i-1}, \mathcal{F}_i), \quad \text{where } \mathcal{F}_i \subseteq \mathcal{F},\ \mathcal{F}_i = \sigma(q, u_{\text{vis}}, \mathbf{s}_{\leqslant i-1}), \tag{2}$$

  where $\sigma(\cdot)$ is a prefix-aware feature selection policy. Static injection is recovered as the special case $\mathcal{F}_i \equiv \mathcal{F}_{\text{stat}}$ for all i.

3 Limitations of Static Feature Injection

Having defined static and dynamic feature injection in GR, we now analyze why the static scheme adopted by most existing GR methods is insufficient (Rajput et al., 2023; Zhou et al., 2026; Zhang et al., 2026a; Chen et al., 2026a). Motivated by the observed cost–gain imbalance of static injection at scale (Zhou et al., 2026), we compare a shared feature-free baseline with feature-specific static-injection models, where each model statically encodes one feature type into the input.

We begin with an **empirical observation**. As shown in Figure 1, increasing the amount of features injected statically, instantiated by user behavior sequences, leads to diminishing or even negative improvements for training, while increasing serving cost. This suggests that indiscriminate, static feature injection is neither consistently beneficial nor cost-efficient. Moving beyond static injection, dynamic injection raises two key challenges: *when* a feature should be incorporated across requests, and *where* in the SID hierarchy it should take effect.

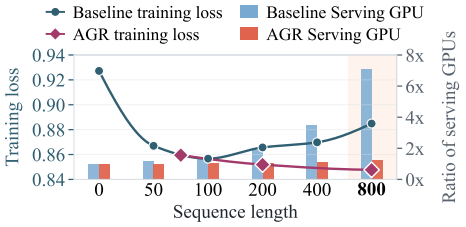


Figure 1: Feature injection efficiency on training loss and serving cost.

For **the first challenge**, static injection assumes an injected feature is universally useful, rather than request-dependent. To diagnose this issue, we construct two test sets: feature-consistent set Fea., where the request and target item share the same feature value, and feature-conflicting set Comp., where their values differ. We measure the injection effect by the perplexity (Jelinek et al., 1977):

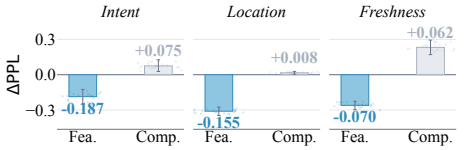


Figure 2: Static injection on feature-aligned (Fea.) and complementary (Comp.) subsets.

$$\Delta\mathrm{PPL} = \mathrm{PPL}_{\text{static}} - \mathrm{PPL}_{\text{w/o feature}}, \quad \mathrm{PPL} = \exp\left(-\frac{1}{L}\sum_{i=1}^{L}\log \mathbb{P}(s_i \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i-1})\right). \quad (3)$$

As shown in Figure 2, the static injection model lowers PPL in Fea. set but raises it in Comp. set, confirming that feature utility is request-dependent. Therefore, efficient feature injection should be request-adaptive, using features only when beneficial.

For **the second challenge**, static injection exposes all decoding steps to the same features. However, RQ-based SIDs typically exhibit coarse-to-fine semantics, suggesting that different steps may require different features. To verify this, we identify, for each feature, the depth i^* where injecting it would most improve next-token prediction (detailed in Section 4.2), and then measure whether static injection actually concentrates its effect at the depth i^*. Concretely, we compare the golden-token probability ratio at each step:

Table 1: Golden-token probability ratios under static feature injection.

Illustration	Feature	$r_{i=i^*}$	$r_{i\neq i^*}$
	Intent	1.032	1.040
	Location	1.003	1.001
	Freshness	1.008	1.010

$$r_i = \frac{\mathbb{P}_{\text{static}}(s_{i+1} \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i})}{\mathbb{P}_{\text{w/o feature}}(s_{i+1} \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i})}, \quad (4)$$

where $\mathbb{P}_{\text{static}}$ and $\mathbb{P}_{\text{w/o feature}}$ denote the golden-token probability produced by models with features statically injected into the model input and without any encoded features, respectively. As shown in Table 1, the probability ratio r_{i^*} is not consistently higher than at other levels. This confirms that static injection spreads its influence uniformly rather than concentrating it where the feature is most needed. Efficient feature injection therefore requires a level-adaptive mechanism that activates each feature only at the decoding depths where it is expected to be informative.

4 Methodology

To enable on-demand feature interaction, we propose *Agentic Generative Retrieval (AGR)*. Section 4.1 introduces the inference framework and tool implementation; Section 4.2 describes how data are constructed and used to operationalize this tool-augmented generative retrieval paradigm.

4.1 AGENTIC GENERATIVE RETRIEVAL

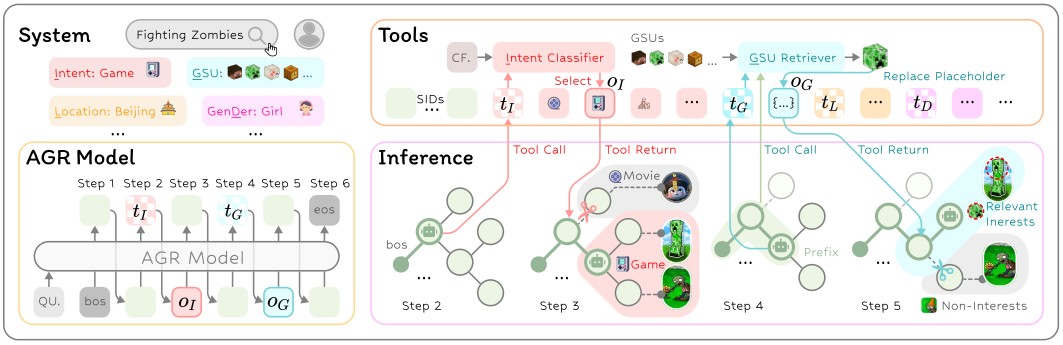


Figure 3: Overview of *AGR*: (1) *AGR Tool* extends the vocabulary with tool and observation tokens, mapping tool calls to feature values; CF. are categorical features; GSU (General Search Unit) are user behavioral sequences. (2) *AGR Model* performs SID generation by predicting tool tokens and adaptively injecting external features via observation tokens (QU: query text and user features). (3) *AGR Inference* enables model-tool interaction for each beam candidate during decoding, forming tool-adaptive retrieval.

***AGR* Framework.** To provide a flexible and extensible interface for feature interaction, we model tool calls and their results as tokens. Specifically, each tool token $t \in \mathcal{T}$ triggers a function call $f_t(q, u, \mathbf{s}_{\leqslant i})$, whose result is represented as an observation token $o_t \in \mathcal{O}$. By extending the vocabulary to $\mathcal{V} = \mathcal{S} \cup \mathcal{T} \cup \mathcal{O}$, the model naturally unifies SID generation and tool invocation into a single autoregressive process. As shown in Figure 3, given the request q, user u_{vis}, decoded prefix $\mathbf{s}_{\leqslant i}$, and prior observations, the model predicts either the next SID token $s_{i+1} \in \mathcal{S}$ or a tool token $t \in \mathcal{T}$ (e.g. t_{I} or t_{G}). Specifically, tool tokens serve solely as decision triggers and are not fed back into the autoregressive context; only the returned observation token (e.g., o_{I} or o_{G}) is injected, with distinct positional encodings to differentiate it from $\mathcal{S}$. As a result, the decoder consumes a single token per step, yet still allowing observations to prune irrelevant branches during generation (e.g., the "Movie" branch in Figure 3). For more details, refer to Algorithm 2 and Example 1.

Tool Implementations. Based on the type of observation returned, tools fall into two categories: *structured feature tools*, and *sequential feature tools*:

(a) *Structured feature tools* return categorical values and can be directly encoded into a single learnable observation token — e.g., a game intent maps to o_{I}:Game, a Beijing location to o_{L}:Beijing, and a high freshness requirement to o_{F}:High. Due to the hierarchical structure of RQ, items with similar features naturally share SID prefixes; for example, videos about games tend to cluster under the same prefixes (Rajput et al., 2023). Thus, these learnable observation tokens could serve as informative signals during decoding.

(b) *Sequential feature tools* return user behavioral sequences (e.g., General Search Unit, GSU (Zhou et al., 2026)), which need to be encoded into a single observation token o_{G}. Various methods could accomplish this (Cao et al., 2026); we adopt two complementary approaches: *Vote*, a parameter-free approach that is efficient for training and inference, and *Attn*, which offers stronger user preference modeling capacity. As shown in Figure 4, *Vote* performs prefix matching: it selects items whose SID prefix matches the decoded one, then picks the most frequent next SID token (i.e., `s3_7`) via majority voting. *Attn* instead operates in the embedding space: it computes the accumulated residual representation of the decoded prefix (e.g., the sum of the embeddings of `s1_1` and `s2_3`), retrieves the Top-E ($E = 5$ for evaluation) most similar items from the sequence, and encodes their next SID token representations into a single embedding via multi-head attention. Unless otherwise stated, we use the *Vote* variant (Table 14).

4.2 TOOL-AUGMENTED *AGR* FINE-TUNING

Data Construction. Having incorporated tool calls and observations into the token vocabulary (Section 4.1), we now describe how to construct training data that teaches the model to invoke tools

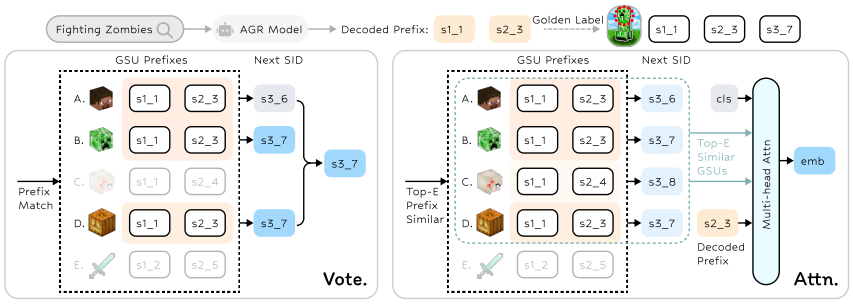


Figure 4: Two variants of the sequential feature tool.

during decoding. Concretely, for each positive sample x^+ with target SIDs $\mathbf{s}^+ = [s_1, \ldots, s_L]$, we augment the sequence by inserting a tool token t at depth i^* and its corresponding observation o_t into the shifted input, so that the model predicts $[s_1, \ldots, s_{i^*}, t, s_{i^*+1}, \ldots, s_L, \texttt{eos}]$ from $[\texttt{bos}, s_1, \ldots, s_{i^*}, o_t, s_{i^*+1}, \ldots, s_L]$. As outlined in Section 3, this construction requires answering two questions: *when* — which samples should be augmented with which tools, and *where* — at which decoding level $1 \leqslant i^* < L$ the tool should be invoked.

- **"When" is a feature applicable?** From the perspective of individual samples, tools are not universally applicable across all scenarios. Leveraging available supervised data, we identify when a tool is applicable and construct $\mathcal{P}(x^+)$, the set of applicable tool-position pairs (t, i) for each positive sample x^+.
  (a) For a *structured feature tool* t, $o_t = f_t(q, u)$ extracts the feature from the request side (e.g., query and all user context) to produce observation o_t, and $g_t(x^+)$ extracts the same feature from the positive sample x^+. To ensure both relevance and non-redundancy, we retain t only when the two extractions agree, and the feature is not already explicit in the context:
  $$\mathcal{P}_{\text{str}}(x^+) = \{(t, i) \mid o_t = g_t(x^+) \wedge \ o_t \text{ absent from model inputs}, \ 1 \leqslant i < L\}. \tag{5}$$
  (b) For a *sequential feature tool* t, $o_{t,i} = f_t(q, u, \mathbf{s}^+_{\leqslant i})$ derives its observation from user behavior sequences, while $g_t(x^+, \mathbf{s}^+_{\leqslant i}) = s^+_{i+1}$ directly extracts the golden next SID. We instantiate f_t with the *Vote* variant f_{vote}. Similarly, we retain t only when the two extractions agree:
  $$\mathcal{P}_{\text{seq}}(x^+) = \left\{(t, i) \mid o_{t,i} = s^+_{i+1}, \ 1 \leqslant i < L\right\}. \tag{6}$$
  To make the model robust to ineffective tool calls, we also sample incorrect predictions, maintaining a voting hit rate of τ to control the hit-to-miss ratio.
- **"Where" to insert the feature?** Having identified the samples amenable to tool insertion, we now determine the optimal depth i^* to insert tools. The key intuition is that a feature should be injected at the depth where it best disambiguates the next SID token. We formalize this via ρ_i, which reflects how discriminative the feature is at that prefix. The depth where ρ changes most sharply is selected as the most valuable position for feature injection:
  $$i^*_t(x^+) = \arg\max_i \left[\rho_{i+1}(o_t) - \rho_i(o_t)\right] \quad \text{s.t. } (t, i) \in \mathcal{P}(x^+), \tag{7}$$
  where $\mathcal{P} = \mathcal{P}_{\text{str}}$ for structured feature tools and $\mathcal{P} = \mathcal{P}_{\text{seq}}$ for sequential feature tools. We define ρ_i for each tool type as follows:
  (a) For a *structured feature tool*, the observation is a discrete attribute of items. To identify where injecting this feature narrows the candidate set most, therefore best disambiguates the next SID token, we compute the log-proportion of items under prefix $\mathbf{s}^+_{\leqslant i}$ that share the same attribute value over the corpus $\mathcal{X}$:
  $$\rho_i(o_t) = \log \mathbb{P}_{x' \sim \mathcal{X}} \left(g_t(x') = o_t \mid \mathbf{s}^+_{\leqslant i}\right). \tag{8}$$
  The sharpest increase in ρ_i identifies the depth where the feature becomes most discriminative, therefore provides the strongest signal for predicting the next SID.
  (b) For a *sequential feature tool*, the observation is the next SID token voted from behavior sequences. To measure where this observation narrows the candidate set most, we compute the log-proportion of items under prefix $\mathbf{s}^+_{\leqslant i}$ that share the same next token over $\mathcal{D}_{\text{train}}$:

$$\rho_i(o_{t,i}) = \log \mathbb{P}_{x' \sim \mathcal{D}_{\text{train}}} \left(g_t(x', \mathbf{s}'_{\leqslant i}) = o_{t,i} \mid \mathbf{s}^+_{\leqslant i} \right). \tag{9}$$

By Theorem 3, maximizing the increase in ρ_i is equivalent to maximizing the model's prediction gain (i.e., $r_i = \mathbb{P}_\theta / \mathbb{P}_{\text{w/o features}}$, analogous to Eq. 4); both are further equivalent to minimizing ρ_i itself. Accordingly, for sequential feature placement, we typically adopt the equivalent criterion $i^* = \arg\min_i \rho_i(o_{t,i})$. Intuitively, a lower ρ_i indicates that the golden SID is rarer among items sharing the same prefix, making the observation more informative for identifying the user's fine-grained or niche preferences.

The "when" and "where" decisions are illustrated in Example 2.

Two-Stage Training. To decouple semantic learning from tool integration for better extensibility, *AGR* training consists of two stages. **The first stage** leverages GR (SID-only) data to learn SID generation following traditional GR methods (Chen et al., 2025c). **The second stage** then introduces *AGR* data, where the model learns to integrate tool calls into generation. The loss now covers both SID tokens and tool tokens. In practice, since tool tokens are unseen during the first stage, their token-level losses are initially much higher than those of SID tokens, risking destabilization of already-learned representations. To mitigate this imbalance, we scale the loss of each tool token by a factor α_k that adapts per training step. Formally, let $\mathbf{v}_{\text{out}}(x) = [v_1, \ldots, v_{|\mathbf{v}_{\text{out}}|}]$ denote the decoder output sequence for the given sample x. For mini-batch $\mathcal{B}_k$ at step k, we measure the prediction difficulty gap between tool and SID tokens: $\beta_k = \frac{\mathbb{E}_{(x,j)\sim\mathcal{B}_k}[\log \mathbb{P}_\theta(v_j | q, u_{\text{vis}}, \mathbf{s}_{\leqslant j-1}) | v_j \notin \mathcal{T}]}{\mathbb{E}_{(x,j)\sim\mathcal{B}_k}[\log \mathbb{P}_\theta(v_j | q, u_{\text{vis}}, \mathbf{s}_{\leqslant j-1}) | v_j \in \mathcal{T}]}$ and clip it to $[\alpha_{\min}, \alpha_{\max}]$ for stability $\alpha_k = \mathbf{1}[v_j \in \mathcal{T}] \cdot \text{clip}(\beta_k, \alpha_{\min}, \alpha_{\max}) + \mathbf{1}[v_j \notin \mathcal{T}]$. The final objective is then delivered as:

$$\mathcal{L}_{\text{AGR}} = -\frac{1}{|\mathcal{B}_k|} \sum_{x \in B_k} \frac{1}{|\mathbf{v}_{\text{out}}|} \sum_{j=1}^{|\mathbf{v}_{\text{out}}|} \alpha_k \cdot \log \mathbb{P}_\theta(v_j \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant j-1}). \tag{10}$$

5 Experiments

5.1 Experimental Setup

Datasets. We evaluate *AGR* on four datasets: Industry, Qilin, JDSearch, and KuaiSAR (large). Industry is collected from real-world user behavior logs, with approximately 300 million records per day; we use June 14–20 for the first training stage, June 21–22 for the second training stage, and June 23 for testing. Qilin is a multimodal retrieval benchmark with app-level user sessions (Chen et al., 2025b); JDSearch focuses on personalized product-search benchmark with real queries and user–item interactions (Liu et al., 2023); and KuaiSAR provides a unified search-and-recommendation benchmark (Sun et al., 2023b). Further details are provided in Table 8.

***AGR* Implementations.** We adopt a 12-layer encoder-decoder BART (Lewis et al., 2020) as the backbone ($\approx$ 574M parameters, including embeddings for approximately 19,000 predefined SID codewords) and construct hierarchical SIDs via RQ-KMeans (Rajput et al., 2023). Dataset-specific SID configurations and tool sets are listed in Table 9; inference follows Algorithm 2. On the large-scale Industry dataset, we train the model sequentially in two stages, with each stage running for one epoch over its corresponding aggregated training data. For the smaller public datasets, we fine-tune with each method for 10 epochs until convergence. For fair comparison, within each dataset, all methods share the same text-pretrained backbone, SID codebooks, training data, learning rate, and hardware configuration. Additional details are provided in Appendix A.

Evaluation Metrics. We adopt Recall@K and Mean Reciprocal Rank (MRR@K) with $K \in \{32, 50, 64\}$ ($K = 50$ for online A/B test). Given N test requests, for i-th query, let $\mathcal{G}_i$ denote the ground-truth item set, $\mathcal{R}_i$ the top-K retrieved items, and r_i the rank of the first ground-truth item ($r_i = \infty$ if none appears in the top-K). Formally, we have:

$$\text{Recall@}K = \frac{1}{N} \sum_{i=1}^{N} \frac{|\mathcal{G}_i \cap \mathcal{R}_i|}{|\mathcal{G}_i|} \quad \text{and} \quad \text{MRR@}K = \frac{1}{N} \sum_{i=1}^{N} \frac{\mathbb{I}[r_i \leqslant K]}{r_i}. \tag{11}$$

5.2 Main Results

As shown in Table 2, ***AGR* consistently outperforms all baselines across all datasets and evaluation metrics.** We compare *AGR* with five generative retrieval baselines: TIGER (Rajput et al., 2023) for standard GR baseline; TIGER Distill Teacher, which first trains a teacher model with up to 400 GSU tokens (Table 4) and then distills its predictions into TIGER using an auxiliary distillation loss with a weight of 0.1; UniSearch (Chen et al., 2025c) for unified short-video retrieval; and GLASS (Cao et al., 2026) and GEMs (Zhou et al., 2026) for long-history modeling. *AGR* achieves the best Recall and MRR at all cutoffs, with relative improvements over TIGER ranging from 6.4% to 14.0% in Recall and from 5.2% to 19.7% in MRR. Its advantages over TIGER Distill Teacher and UniSearch demonstrate the benefit of dynamically acquiring request-specific evidence during decoding, a capability absent from these methods. Its advantages over GLASS and GEMs further demonstrate that adaptive tool integration exploits long-term behavioral features more effectively than direct sequence modeling.

Table 2: Overall performance across four datasets. Best and second-best results are bolded and marked with * respectively; "*Relative gain*" denotes the relative improvement of *AGR* over TIGER. Statistical significance is assessed via paired bootstrap tests over request-level outcomes. We construct 1,000 bootstrap resamples with replacement (10,000 requests for Industry from Jun. 23 held-out data; 5,000 for each public benchmark), with the same sampled indices across methods. All reported gains of *AGR* are significant at $p < 0.05$.

Dataset	Method	Recall@K			MRR@K		
		Recall@64	Recall@50	Recall@32	MRR@64	MRR@50	MRR@32
Industry	TIGER (Rajput et al., 2023)	0.2957	0.2863	0.2692	0.2257	0.2254	0.2253
	TIGER Distill Teacher	0.3278	0.3169	0.2987	0.2528	0.2524	0.2521
	UniSearch (Chen et al., 2025c)	0.3282	0.3153	0.2979	0.2566	0.2563	0.2559
	GLASS (Cao et al., 2026)	0.3289	0.3160	0.2986	0.2573	0.2570	0.2566
	GEMs (Zhou et al., 2026)	0.3292*	0.3165*	0.2990*	0.2639*	0.2635*	0.2632*
	AGR	**0.3368**	**0.3264**	**0.3070**	**0.2701**	**0.2699**	**0.2693**
	Relative gain over TIGER	+13.90%	+14.01%	+14.04%	+19.67%	+19.74%	+19.53%
Qilin	TIGER (Rajput et al., 2023)	0.3330	0.3115	0.2748	0.2923	0.2917	0.2912
	TIGER Distill Teacher	0.3407	0.3198	0.2811	0.3002	0.2999	0.2990
	UniSearch (Chen et al., 2025c)	0.3441	0.3236	0.2838	0.3038	0.3036	0.3025
	GLASS (Cao et al., 2026)	0.3448	0.3243	0.2845	0.3045	0.3043	0.3032
	GEMs (Zhou et al., 2026)	0.3467*	0.3267*	0.2874*	0.3071*	0.3068*	0.3061*
	AGR	**0.3551**	**0.3324**	**0.2924**	**0.3076**	**0.3069**	**0.3065**
	Relative gain over TIGER	+6.64%	+6.71%	+6.40%	+5.23%	+5.21%	+5.25%
JDSearch	TIGER (Rajput et al., 2023)	0.5277	0.5097	0.4749	0.3092	0.3089	0.3080
	TIGER Distill Teacher	0.5360	0.5243	0.4936	0.3292	0.3227	0.3087
	UniSearch (Chen et al., 2025c)	0.5398	0.5315	0.5029	0.3392	0.3295	0.3083
	GLASS (Cao et al., 2026)	0.5405	0.5322	0.5036	0.3399*	0.3302	0.3090
	GEMs (Zhou et al., 2026)	0.5663*	0.5440*	0.5079*	0.3338	0.3335*	0.3326*
	AGR	**0.5670**	**0.5521**	**0.5163**	**0.3433**	**0.3430**	**0.3422**
	Relative gain over TIGER	+7.45%	+8.32%	+8.72%	+11.03%	+11.04%	+11.10%
KuaiSAR	TIGER (Rajput et al., 2023)	0.4327	0.4179	0.3953	0.3132	0.3009	0.2990
	TIGER Distill Teacher	0.4448	0.4356	0.4169	0.3332	0.3173	0.3065
	UniSearch (Chen et al., 2025c)	0.4506	0.4444	0.4279	0.3432	0.3255	0.3098
	GLASS (Cao et al., 2026)	0.4513	0.4451	0.4286	0.3439	0.3262	0.3105
	GEMs (Zhou et al., 2026)	0.4655*	0.4483*	0.4293*	0.3447*	0.3312*	0.3288*
	AGR	**0.4785**	**0.4582**	**0.4335**	**0.3507**	**0.3368**	**0.3348**
	Relative gain over TIGER	+10.58%	+9.64%	+9.66%	+11.97%	+11.93%	+11.97%

5.3 Ablation Study

Our ablation and analysis primarily focus on the Industry dataset, as it supports more diverse tool construction than existing public benchmarks. Additional ablations and analyses are in Section C.

Adaptive feature injection outperforms predetermined strategies. A defining principle of *AGR* is that feature injection should be driven by the model's generation state rather than predetermined. We ablate this across four strategies over a feature set including intent, location, freshness and GSU sequence. *Static feature injection* feeds all features into the encoder without selection or weighting. *Query-guided gating* uses a lightweight MLP,

Table 3: Varying feature interaction strategies.

Strategies	Recall@50
(a) Static feature injection	0.3159
(b) Query-guided gating	0.3161
(c) Fixed-position tool injection	0.3174
(d) *AGR* (ours)	**0.3264**

whose parameter count is substantially smaller (≈4,100 parameters) than that of the backbone, to learn query-dependent feature weights. *Fixed-position tool injection* retrieves features via tool calls, but at a predetermined position. *AGR* uses the same tool-call mechanism but injects adaptively. As shown in Table 3, query-guided gating and fixed-position injection both outperform static injection, yet *AGR* achieves the best Recall@50, confirming the effectiveness of adaptive injection.

Sequential feature tools match long sequence encoding at a fraction of the cost. Tool-based retrieval compresses sequential features into discrete observations to keep serving cost manageable. We quantify this trade-off by comparing *AGR* with directly appending up to 400 GSU tokens to the input. As shown in Table 4, direct encoding improves Recall@50 from 0.2863 to 0.3241, but at steep cost: training time rises from 1.163 to 10.794 seconds per step, and serving cost rises from 2.38 to 8.33 GPUs per 1K requests. In contrast, *AGR* achieves comparable Recall@50 of 0.3237, while reducing training time and serving cost by 81.8% and 42.9%. Although *AGR* roughly doubles the serving cost over the no-feature baseline, it remains far more efficient than direct encoding. We also observe a similar trade-off for structured features (Appendix C).

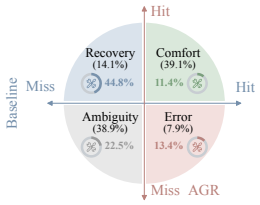
Table 4: Comparison of direct GSU sequence encoding and dynamic acquisition in *AGR*. Training time is measured under identical batch size and configuration. Serving cost is the number of GPUs required per 1K requests under the same offered load and GPU configuration.

Method	Recall@50	Train Time (sec/step)	Serving Cost (GPUs/1K req.)
Baseline	0.2863	1.163	2.38
+ 50 relevant GSU tokens	0.3195	2.413	2.70
+ 100 relevant GSU tokens	0.3202	3.671	3.03
+ 400 relevant GSU tokens	0.3241	10.794	8.33
***AGR* w/ GSU tool only**	**0.3237**	**1.969**	**4.76**

5.4 Experimental Analysis

***AGR* recovers baseline failures through selective, precise tool usage.** To trace the source of *AGR*'s gains, we partition 10,000 Industry test instances into four quadrants by comparing *AGR* against the tool-free baseline: **C**omfort (both hit), **A**mbiguity (both miss), **R**ecovery (only *AGR* hits), and **E**rror (only baseline hits). As shown in Table 5, *Recovery* accounts for 14.1% of instances versus 7.9% for *Error*, yielding a net improvement of +6.2pp in Recall@50. Notably, tool invocation is heavily concentrated in *Recovery*, where the tool-call rate reaches 44.8%, far above the 20.6% overall average. Within *Recovery*, we further compare tool-invoking and no-tool beams among the most confident top-10 beams for each query: tool-invoking beams achieve a 15.1% hit rate versus 3.7% for no-tool beams (4.1×), and every tool type outperforms the no-tool beams. These findings confirm that *AGR* selectively invokes tools on queries the baseline fails, and that tool-assisted beams are substantially more likely to retrieve the correct item.

Table 5: CARE analysis and tool usage in *Recovery*. Percentages in parentheses show case proportions.

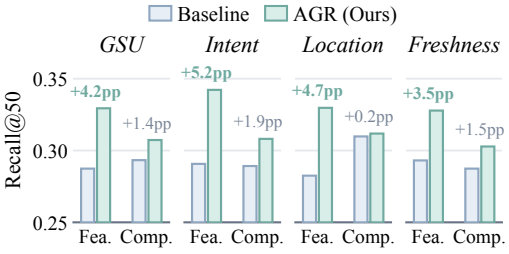


Type	Beams (%)	Hits	Hit Rate
w/ tool call	6,304 (44.8%)	952	**15.1%**
GSU	5,664 (40.2%)	888	15.7%
Intent	304 (2.2%)	38	12.5%
Freshness	184 (1.3%)	15	8.2%
Location	152 (1.1%)	11	7.2%
w/o tool call	7,776 (55.2%)	284	**3.7%**
Top-10 beams	14,080	1,236	8.8%

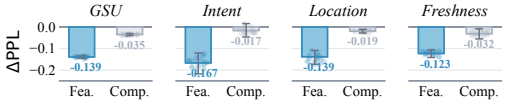
***AGR* yields disproportionately larger gains where features matter most.** To examine whether *AGR*'s improvements are tied to genuine feature relevance we partition the data into Fea. and Comp. subsets based on whether the target item is characterized by the corresponding feature (Section 3). As shown in Figure 5(a), *AGR* achieves more than twice the recall gain on Fea. compared to Comp., confirming that its improvements concentrate on feature-dependent queries rather than arising from uniform score shifts.

***AGR* injects evidence at the right moment, avoiding the degradation caused by ill-timed feature injection.** To examine whether *AGR* resolves the challenge of "**when**" in static injection, we report $\Delta\text{PPL} = \text{PPL}_{\text{AGR}} - \text{PPL}_{\text{w/o feature}}$ (analogous to Eq. 3) on the Fea. and Comp. set. As shown in Figure 2, static injection lowers PPL on Fea. but *raises* it on Comp., exposing its inability to sense when feature injection is actually needed. *AGR*, by contrast, yields negative ΔPPL on *both* sets (Figure 5(b)), with a consistently larger reduction on Fea., confirming that it activates tool calls only when evidence is beneficial and withholds them otherwise.

***AGR* concentrates feature signals at the most informative positions rather than spreading them indiscriminately.** To examine whether *AGR* resolves the challenge of "**where**" in static injection, we let $i^* = \arg\max_i \Delta\rho_i$ denote the SID level with the largest change in feature concentration and measure the golden-token probability ratio (Eq. 4) at i^* relative to other levels. As shown in Figure 5(c), static injection produces nearly the same ratio at i^* and elsewhere, whereas *AGR* consistently yields a larger ratio at i^* for every feature, with the contrast strongest for Location (1.116 vs. 1.012) and Freshness (1.218 vs. 1.063), confirming that *AGR* allocates feature influence precisely where it carries the most discriminative power.



(a) Comparison between *AGR* and baseline on feature-aligned (Fea.) and complementary (Comp.) subsets.



(b) *AGR* perplexity relative to Baseline,

Feature	Static injection		*AGR*	
	$r_{i=i^*}$	$r_{i\neq i^*}$	$r_{i=i^*}$	$r_{i\neq i^*}$
GSU	1.017	1.009	**1.041** ↑	1.012
Intent	1.032	1.040	**1.056** ↑	1.050
Location	1.003	1.001	**1.116** ↑	1.012
Freshness	1.008	1.010	**1.218** ↑	1.063

(c) Comparison of golden-token probability ratio.

Figure 5: Performance analysis of *AGR* under different feature requirements.

5.5 Online A/B Test

We deploy *AGR* in our production search system with an average serving latency of 35ms, well within our online budget. We then conduct a live experiment consisting of a 7-day A/A test followed by a 7-day A/B test, with the production baseline and *AGR* each serving 10% of randomly sampled user traffic. As shown in Table 6, *AGR* yields consistent gains along three axes of user experience. *Relevance*, defined as the fraction of PVs (i.e., search requests) whose top-4 results are strongly relevant to the query, improves by +0.205%, indicating that *AGR* retrieves more query-relevant items. *Query reformulation rate* drops by -0.690%, showing that users are more often satisfied on the first query and no longer need to rephrase it. Finally, *search penetration rate*, defined as the fraction of users who engage with search, rises by +0.06%. As an upstream user-behavior metric that *AGR* does not optimize directly, its improvement suggests that better single-query experience propagates into higher willingness to use search.

Table 6: Online A/B test results.

Metric	Change	95% CI
Relevance ↑	+0.205%	[+0.12%, +0.28%]
Query Reformulation Rate ↓	−0.690%	[−1.13%, −0.20%]
Search Penetration Rate ↑	+0.060%	[+0.03%, +0.14%]

6 Limitations and Future Work

AGR interleaves tool invocation with SID generation, but has three limitations. First, its training data are constructed using predefined rules rather than learned policies; future work could apply agentic reinforcement learning to optimize tool use under the sparse rewards common in search and recommendation. Second, the current tool ecosystem is limited and should be extended to more tools and retrieval domains. Third, trajectories are constructed along golden SID prefixes using target-side agreement, creating a gap from free-running inference. Optimizing and evaluating tool decisions under model-generated prefixes is therefore an important future direction.

7 Conclusion

We presented *Agentic Generative Retrieval (AGR)*, a tool-augmented paradigm that selectively acquires and incorporates heterogeneous features during hierarchical SID generation. *AGR* determines when, and where features should be introduced based on the request and decoded SID prefix. We train this capability using tool-augmented data constructed from feature applicability and discriminability. Evaluations on an industrial dataset and public benchmarks, together with an online A/B test, demonstrate the effectiveness and efficiency of *AGR*, highlighting tool-augmented generation as a promising direction for large-scale generative retrieval.

AI Assistance Statement

In this work, we used generative AI tools to aid in writing and improve the readability of the manuscript, and to assist with searching for and identifying relevant literature. We have not used generative AI tools to generate synthetic datasets, formulate or prove mathematical claims, design or implement core methods, or interpret experimental results. We have reviewed all AI-assisted content for accuracy and correctness, and we take full responsibility for the final content of this paper, including all text, claims, and experimental results.

Ethics Statement

Our evaluations use the Industry dataset and public benchmarks, all of which are anonymized and processed without personally identifiable information. Although *AGR* improves feature utilization in retrieval systems, its deployment should carefully monitor potential biases and ensure the responsible use of user-related features.

Reproducibility Statement

All data, algorithms, and experimental configurations used in this study are described in sufficient detail in the paper to support reproducibility. Due to copyright and privacy considerations, the implementation and industrial data are not publicly available at this time. We are conducting the necessary internal reviews and intend to release, where permitted, the core algorithms, appropriately anonymized data, and experimental details at `https://anonymous.4open.science/r/AGR-EF62/`.

References

Vladimir Baikalov, Iskander Bagautdinov, and Sergey Muravyov. Mitigating collaborative semantic id staleness in generative retrieval. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3602–3608, 2026.

Michele Bevilacqua, Giuseppe Ottaviano, Patrick Lewis, Scott Yih, Sebastian Riedel, and Fabio Petroni. Autoregressive search engines: Generating substrings as document identifiers. *Advances in Neural Information Processing Systems*, 35:31668–31683, 2022.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. In *9th International Conference on Learning Representations*, 2021.

Shiteng Cao, Junda She, Ji Liu, Bin Zeng, Chengcheng Guo, Kuo Cai, Qiang Luo, Ruiming Tang, Han Li, Kun Gai, et al. Glass: A generative recommender for long-sequence modeling via sid-tier and semantic search. *arXiv preprint arXiv:2602.05663*, 2026.

Ben Chen, Xian Guo, Siyuan Wang, Zihan Liang, Yue Lv, Yufei Ma, Xinlong Xiao, Bowen Xue, Xuxin Zhang, Ying Yang, et al. Onesearch: A preliminary exploration of the unified end-to-end generative framework for e-commerce search. *arXiv preprint arXiv:2509.03236*, 2025a.

Ben Chen, Siyuan Wang, Yufei Ma, Zihan Liang, Xuxin Zhang, Yue Lv, Ying Yang, Huangyu Dai, Lingtao Mao, Tong Zhao, et al. Onesearch-v2: The latent reasoning enhanced self-distillation generative search framework. *arXiv preprint arXiv:2603.24422*, 2026a.

Jia Chen, Qian Dong, Haitao Li, Xiaohui He, Yan Gao, Shaosheng Cao, Yi Wu, Ping Yang, Chen Xu, Yao Hu, et al. Qilin: A multimodal information retrieval dataset with app-level user sessions. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 3670–3680, 2025b.

Jiahui Chen, Xiaoze Jiang, Zhibo Wang, Quanzhi Zhu, Junyao Zhao, Feng Hu, Kang Pan, Ao Xie, Maohua Pei, Zhiheng Qin, et al. Unisearch: Rethinking search system with a unified generative architecture. *arXiv preprint arXiv:2509.06887*, 2025c.

Zhiguo Chen, Guohao Sun, Yiming Qiu, Xingzhi Yao, Mingming Li, Huimu Wang, Yangqi Zhang, Songlin Wang, and Sulong Xu. Rad-dpo: Robust adaptive denoising direct preference optimization for generative retrieval in e-commerce. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4507–4512, 2026b.

Jiaxin Deng, Shiyao Wang, Kuo Cai, Lejian Ren, Qigen Hu, Weifeng Ding, Qiang Luo, and Guorui Zhou. Onerec: Unifying retrieve and rank with generative recommender and iterative preference alignment. *arXiv preprint arXiv:2502.18965*, 2025.

Songtao Fang, Zihao Xu, Shaowei Wei, Jin Zhang, and Zhuojun Wang. Generative retrieval for e? commerce: Jointly learning embedding and codebook with same product cluster. In *Proceedings of the ACM Web Conference 2026*, pp. 8369–8372, 2026.

Kairui Fu, Changfa Wu, Kun Yuan, Binbin Cao, Dunxian Huang, Yuliang Yan, Junjun Zheng, Jianning Zhang, Silu Zhou, Jian Wu, et al. Rankgr: Rank-enhanced generative retrieval with listwise direct preference optimization in recommendation. *arXiv preprint arXiv:2602.08575*, 2026a.

Kairui Fu, Tao Zhang, Shuwen Xiao, Ziyang Wang, Xinming Zhang, Chenchi Zhang, Yuliang Yan, Junjun Zheng, Xiangheng Kong, Shengyu Zhang, et al. Forge: Forming semantic identifiers for generative retrieval in industrial datasets. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 8858–8869, 2026b.

Chengcheng Guo, Kuo Cai, Yu Zhou, Qiang Luo, Ruiming Tang, Han Li, Kun Gai, and Guorui Zhou. Promise: Process reward models unlock test-time scaling laws in generative recommendations. *arXiv preprint arXiv:2601.04674*, 2026.

Yupeng Hou, Haven Kim, Clark Mingxuan Ju, Eduardo Escoto, Neil Shah, and Julian McAuley. Expressiveness limits of autoregressive semantic id generation in generative recommendation. *arXiv preprint arXiv:2605.06331*, 2026.

Fred Jelinek, Robert L Mercer, Lalit R Bahl, and James K Baker. Perplexity—a measure of the difficulty of speech recognition tasks. *The journal of the Acoustical Society of America*, 62(S1): S63–S63, 1977.

Jie Jiang, Yang Wu, Qian Li, Yuling Xiong, Hongbo Tang, Xun Liu, Haoze Wang, Jun Zhang, Huan Yu, and Hailong Shi. Scoter: Structured chain-of-thought transfer for enhanced recommendation. *arXiv preprint arXiv:2511.19514*, 2025.

Jie Jiang, Yang Wu, Qian Li, Yuling Xiong, Yihang Su, Junbang Huo, Longfei Lu, Jun Zhang, and Huan Yu. Diffureason: Bridging latent reasoning and generative refinement for sequential recommendation. *arXiv preprint arXiv:2602.09744*, 2026.

Clark Mingxuan Ju, Tong Zhao, Leonardo Neves, Liam Collins, Bhuvesh Kumar, Jiwen Ren, Lili Zhang, Wenfeng Zhuo, Wen Zhang, Xiao Bai, et al. Semantic ids for recommender systems at snapchat: Use cases, technical challenges, and design choices. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 4694–4699, 2026.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 7871–7880, 2020.

Fuchun Li, Qian Li, Xingyu Gao, Bocheng Pan, Yang Wu, Jun Zhang, Huan Yu, Jie Jiang, Jinsheng Xiao, and Hailong Shi. Chainrec: An agentic recommender learning to route tool chains for diverse and evolving interests. *arXiv preprint arXiv:2602.10490*, 2026a.

Mingming Li, Huimu Wang, Zuxu Chen, Guangtao Nie, Yiming Qiu, Guoyu Tang, Lin Liu, and Jingwei Zhuo. Generative retrieval with preference optimization for e-commerce search. *arXiv preprint arXiv:2407.19829*, 2024.

Qinyao Li, Xiaoyang Zheng, Qihang Zhao, Ke Xu, Zhongbo Sun, Chao Wang, Chenyi Lei, Han Li, and Wenwu Ou. Diffusiongs: Generative search with query conditioned diffusion in kuaishou. In *Proceedings of the ACM Web Conference 2026*, pp. 8405–8408, 2026b.

Fake Lin, Binbin Hu, Zhi Zheng, Xi Zhu, Ziqi Liu, Zhiqiang Zhang, Jun Zhou, and Tong Xu. Token-level collaborative alignment for llm-based generative recommendation. In *Proceedings of the ACM Web Conference 2026*, pp. 6909–6919, 2026.

Jiongnan Liu, Zhicheng Dou, Guoyu Tang, and Sulong Xu. Jdsearch: A personalized product search dataset with real queries and full interactions. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2945–2952, 2023.

Zhanyu Liu, Shiyao Wang, Xingmei Wang, Rongzhou Zhang, Jiaxin Deng, Honghui Bao, Jinghao Zhang, Wuchao Li, Pengfei Zheng, Xiangyu Wu, et al. Onerec-think: In-text reasoning for generative recommendation. *arXiv preprint arXiv:2510.11639*, 2025.

Hou-Wan Long, Yicheng Song, Zidong Wang, and Tianshu Sun. Kp-agent: Keyword pruning in sponsored search advertising via llm-powered contextual bandits. In *Proceedings of the 34th ACM International Conference on Information and Knowledge Management*, pp. 4991–4995, 2025.

Wenhao Lu, Jian Jiao, and Ruofei Zhang. Twinbert: Distilling knowledge to twin-structured compressed bert models for large-scale retrieval. In *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, pp. 2645–2652, 2020.

Shashank Rajput, Nikhil Mehta, Anima Singh, Raghunandan Hulikal Keshavan, Trung Vu, Lukasz Heldt, Lichan Hong, Yi Tay, Vinh Tran, Jonah Samost, et al. Recommender systems with generative retrieval. *Advances in Neural Information Processing Systems*, 36:10299–10315, 2023.

Ninglu Shao, Jinshan Wang, Chenxu Wang, Qingbiao Li, and Xiaoxue Zang. Great: Guiding query generation with a trie for recommending related search about video at kuaishou. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 4818–4826, 2025.

Weiwei Sun, Lingyong Yan, Zheng Chen, Shuaiqiang Wang, Haichao Zhu, Pengjie Ren, Zhumin Chen, Dawei Yin, Maarten Rijke, and Zhaochun Ren. Learning to tokenize for generative retrieval. *Advances in Neural Information Processing Systems*, 36:46345–46361, 2023a.

Zhongxiang Sun, Zihua Si, Xiaoxue Zang, Dewei Leng, Yanan Niu, Yang Song, Xiao Zhang, and Jun Xu. Kuaisar: A unified search and recommendation dataset. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 5407–5411, 2023b.

Yi Tay, Vinh Tran, Mostafa Dehghani, Jianmo Ni, Dara Bahri, Harsh Mehta, Zhen Qin, Kai Hui, Zhe Zhao, Jai Gupta, et al. Transformer memory as a differentiable search index. *Advances in Neural Information Processing Systems*, 35:21831–21843, 2022.

OneRec Team, Biao Yang, Boyang Ding, Chenglong Chu, Dunju Zang, Fei Pan, Han Li, Hao Jiang, Honghui Bao, Huanjie Wang, et al. Onereason technical report. *arXiv preprint arXiv:2606.06260*, 2026.

Yejing Wang, Shengyu Zhou, Jinyu Lu, Qidong Liu, Xinhang Li, Wenlin Zhang, Feng Li, Pengjie Wang, Chuan Yu, Jian Xu, et al. Gflowgr: Fine-tuning generative recommendation frameworks with generative flow networks. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1925–1936, 2026a.

Yejing Wang, Shengyu Zhou, Jinyu Lu, Ziwei Liu, Langming Liu, Maolin Wang, Wenlin Zhang, Feng Li, Wenbo Su, Pengjie Wang, Jian Xu, and Xiangyu Zhao. Nezha: A zero-sacrifice and hyperspeed decoding architecture for generative recommendations. In *Proceedings of the ACM Web Conference 2026*, pp. 8073–8082, 2026b.

Yujing Wang, Yingyan Hou, Haonan Wang, Ziming Miao, Shibin Wu, Qi Chen, Yuqing Xia, Chengmin Chi, Guoshuai Zhao, Zheng Liu, et al. A neural corpus indexer for document retrieval. *Advances in Neural Information Processing Systems*, 35:25600–25614, 2022.

Zesheng Wang, Longfei Xu, Weidong Deng, Huimin Yan, Kaikui Liu, and Xiangxiang Chu. Intrr: A framework for integrating sid redistribution and length reduction. *arXiv preprint arXiv:2602.20704*, 2026c.

Zhibo Wang, Xiaoze Jiang, Zhiheng Qin, and Enyun Yu. Personalized query auto-completion for long and short-term interests with adaptive detoxification generation. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.2*, pp. 5018–5028, 2025.

Ao Xie, Jiahui Chen, Quanzhi Zhu, Xiaoze Jiang, Zhiheng Qin, Enyun Yu, and Han Li. Crops: Improving dense retrieval with cross-perspective positive samples in short-video search. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pp. 34061–34069, 2026.

Ben Xue, Dan Liu, Lixiang Wang, Mingjie Sun, Peng Wang, Pengfei Zhang, Shaoyun Shi, Tianyu Xu, Yunhao Sha, Zhiqiang Liu, et al. 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.

Huimin Yan, Longfei Xu, Junjie Sun, Ni Ou, Wei Luo, Xing Tan, Ran Cheng, Kaikui Liu, and Xiangxiang Chu. Intsr: An integrated generative framework for search and recommendation. *arXiv preprint arXiv:2509.21179*, 2025.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tianyi Tang, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. Qwen2.5 technical report. *arXiv preprint arXiv:2412.15115*, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Qiya Yang, Xiaoxi Liang, Zeping Xiao, Ying Cao, YingJie Deng, Yuxin Ren, Yalong Wang, and Yongqi Liu. Breaking the likelihood trap: Consistent generative recommendation with graph-structured model. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 8379–8388, 2026.

Zhongchao Yi, Kai Feng, Xiaojian Ma, Yalong Wang, Yongqi Liu, Han Li, Zhengyang Zhou, and Yang Wang. Dualgr: Generative retrieval with long and short-term interests modeling. In *Proceedings of the ACM Web Conference 2026*, pp. 8329–8332, 2026.

Hansi Zeng, Chen Luo, Bowen Jin, Sheikh Muhammad Sarwar, Tianxin Wei, and Hamed Zamani. Scalable and effective generative information retrieval. In *Proceedings of the ACM Web Conference 2024*, pp. 1441–1452, 2024.

Jiaqi Zhai, Lucy Liao, Xing Liu, Yueming Wang, Rui Li, Xuan Cao, Leon Gao, Zhaojie Gong, Fangda Gu, Michael He, et al. Actions speak louder than words: Trillion-parameter sequential transducers for generative recommendations. *arXiv preprint arXiv:2402.17152*, 2024.

Haobo Zhang, Yutao Zhu, Kelong Mao, Tianhao Li, and Zhicheng Dou. Recthinker: An agentic framework for tool-augmented reasoning in recommendation. *arXiv preprint arXiv:2603.09843*, 2026a.

Kun Zhang, Jingming Zhang, Wei Cheng, Yansong Cheng, Jiaqi Zhang, Hao Lu, Xu Zhang, Haixiang Gan, Jiangxia Cao, Tenglong Wang, et al. Onemall: One architecture, more scenarios–end-to-end generative recommender family at kuaishou e-commerce. *arXiv preprint arXiv:2601.21770*, 2026b.

Kai Zheng, Haijun Zhao, Rui Huang, Beichuan Zhang, Na Mou, Yanan Niu, Yang Song, Hongning Wang, and Kun Gai. Full stage learning to rank: A unified framework for multi-stage systems. In *Proceedings of the ACM Web Conference 2024*, pp. 3621–3631, 2024.

Yu Zhou, Chengcheng Guo, Kuo Cai, Ji Liu, Qiang Luo, Ruiming Tang, Han Li, Kun Gai, and Guorui Zhou. Gems: Breaking the long-sequence barrier in generative recommendation with a multi-stream decoder. *arXiv preprint arXiv:2602.13631*, 2026.

Yanyan Zou, Junbo Qi, Lunsong Huang, Yu Li, Kewei Xu, Jiahao Gao, Binglei Zhao, Xuanhua Yang, Sulong Xu, and Shengjie Li. Genrec: A preference-oriented generative framework for large-scale recommendation. In *Proceedings of the 49th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 5118–5123, 2026.

Table 7: Summary of notations.

Notation	Description
Retrieval setting	
$\mathcal{X}, x$	The item corpus and an item in the corpus.
q, u_{vis}, u	The query, model-visible user context and all user context.
K	The number of items retrieved from $\mathcal{X}$ by beam search.
Tr	The trie constructed from all legal SIDs in the item corpus.
SID generation and scoring	
$\mathbf{s} = [s_1, \ldots, s_L]$	The length-L SID obtained by quantizing $\mathbf{h}_x$, where $s_i \in \mathcal{S}_i$.
$\mathbf{s}_{\leqslant i}$	The clean SID prefix $[s_1, \ldots, s_i]$, obtained by removing tool tokens from $\mathbf{v}_{\text{out}}$.
$\mathbf{v}_{\text{in}}$	The decoder input sequence, which may contain an observation token.
$\mathbf{v}_{\text{out}}$	The model output sequence, which may contain a tool token.
$\mathbb{P}_\theta$	Autoregressive conditional probability distribution parameterized by θ.
$\pi_\theta(\mathbf{s}_{\leqslant i})$	Accumulated log-probability used to score decoded sequences.
$\mathcal{F}$	The candidate feature set from which subsets are selected for injection.
σ	Prefix-aware policy that selects a subset $\mathcal{F}_i \subseteq \mathcal{F}$ at each decoding step.
Tools and observations	
$\mathcal{T}, t$	The set of available tool tokens and a tool identifier.
$\mathcal{O}, o$	The observation-token space and an observation token returned by a tool.
f_t	Tool t, which returns $o_{t,i} = f_t(q, u, \mathbf{s}_{\leqslant i}) \in \mathcal{O}$.
g_t	The item-side attribute extractor.
M	Maximum tool-call quota per beam-decoding path.
Interpretive analysis	
ρ_i	Feature discriminability given prefix $\mathbf{s}_{\leqslant i}$

A *AGR* IMPLEMENTATIONS

This section provides further implementation details of *AGR*. For datasets with textual features (Industry and Qilin (Chen et al., 2025b)), we adopt Qwen2.5 (Yang et al., 2024) as a parameter-shared dual-encoder model and train it with *Query-to-Item (Q2I)* supervision. For anonymous datasets (JD-Search (Liu et al., 2023) and KuaiSAR (Sun et al., 2023b)), we instead use TwinBERT (Lu et al., 2020) to obtain item embeddings. We then apply hierarchical RQ-KMeans to construct their Semantic ID (SID) codebooks. Specifically, for the Industry dataset, which contains billions of recall samples, we use codebook sizes of $(8192, 4096, 2048, 2048, 2048)$, whereas for the comparatively smaller public datasets, we use $(512, 256, 128)$. We next describe the implementation of *AGR* in terms of tools, model architecture, inference, training-data construction, and training procedure.

A.1 DATASETS AND SETTINGS

Table 8 shows the details of the used datasets, Table 9 summarizes the two types of tools and their dataset-specific implementations, and Table 10 summarizes the training and inference settings.

A.2 TOOL IMPLEMENTATIONS

For structured feature tools, each categorical value is represented by a learnable observation embedding, initialized by mean-pooling the pretrained embeddings of the text tokens corresponding to its feature value. For sequential feature tools, historical GSU items are stored with their hierarchical SIDs. The Attn variant uses $E = 5$ retrieved GSU items and an 8-head attention module to aggregate their successor-SID representations. Observation tokens use absolute positional embeddings distinct from those of autoregressive SID tokens, such that an injected observation conditions the next prediction without advancing the SID decoding depth.

A.3 MODEL INFERENCE

Inference interface and adaptive interaction. We instantiate *AGR* with BART, although the method only requires an autoregressive interface and also supports decoder-only backbones such

Table 8: Statistics of Industry and three public benchmarks: Qilin (Chen et al., 2025b), JDSearch (Liu et al., 2023), and KuaiSAR-L (Sun et al., 2023b).

	Retrieval and behavior scale						
Dataset	Users	Items	Queries	Interactions	Coverage and scenario	Content and feedback	Use in *AGR*
Industry	∼2B	∼1B	∼2.1B*	∼2.1B*	Production app search; Jun. 14–23	Search logs with impression, click, and engagement supervision	Qwen2.5 Q2I embeddings; Jun. 14–20 / 21–22 / 23 for Stage I / Stage II / test
Qilin	15,482	1.98M	57,188	2.50M	Xiaohongshu app sessions; search, recommendation, and DQA	Text, 5.01M images, and video notes; six engagement labels	Qwen2.5 Q2I embeddings; click-history, gender, age, and platform tools
JDSearch	173,831	12.87M	171,728†	26.67M	JD.com product search; one-year user histories	Anonymized queries and product metadata; click, cart, follow, and purchase	TwinBERT item embeddings; product-interaction history tool
KuaiSAR-L	25,877	6.89M	453,667	19.66M	Kuaishou search + recommendation; 19 days	Anonymized queries and item features; exposure, click, play, like, follow, and share	TwinBERT item embeddings; short-video viewing-history tool

(1) Notes. B and M denote billion and million; DQA denotes deep query answering; Q2I denotes query-to-item supervision. *Industry contains approximately 300M query-level interaction records per day, hence about 2.1B Stage-I queries/interactions over Jun. 14–20; its user count is of the same order of magnitude. "Interactions" denotes logged query–item or user–item behavior records (e.g., impressions, clicks, or purchases), rather than training actions or tool calls; because event definitions differ, these counts are not directly comparable across datasets. †JDSearch reports the number of test queries. KuaiSAR-L denotes the large release (25,877 users over 19 days), not KuaiSAR-small. ***(2) Data splits and retrieval corpora.*** For Industry, we use Jun. 14–20 for Stage-I training, Jun. 21–22 for Stage-II training, and the held-out data on Jun. 23 for offline evaluation. For statistical testing, we construct 1,000 paired bootstrap resamples from the Jun. 23 test pool, each containing 10,000 requests sampled with replacement. The same sampled request indices are used for all compared methods. For Qilin, we follow its official search-domain split, consisting of 44,024 training instances and 6,192 test instances. Only search-domain instances are used in our retrieval experiments. For JDSearch, we follow its original temporal construction, in which each user's last valid search is used as the request, resulting in 171,728 queries. The preceding interactions are used to construct the training instances and the product-interaction histories available to the sequential feature tool. The target interaction and future behaviors are excluded from the tool state. Then we pick up 10,000 for testing and use remaining requests for training. For KuaiSAR-L, we randomly sample 10,000 requests for testing and use the remaining 443,667 requests for training. For every request, the viewing-history tool accesses only behaviors preceding that request. For statistical testing on each public benchmark, we construct 1,000 paired bootstrap resamples, each containing 5,000 test requests sampled with replacement from the corresponding test set. Identical sampled request indices are used to evaluate all methods within each resample.

Table 9: Dataset-specific tool configurations. SE denotes *Sequential feature tool*, and ST denotes *Structured feature tool*.

Dataset	**Tool**	**Observations**	**Type**
Industry	Short video viewing behaviors (i.e., GSU) retrieval	Successor SID tokens	SE.
	Query intent classification	43 intent categories	ST.
	User location lookup	162 city values	ST.
	Query freshness classification	{high, medium, low, unknown}	ST.
Qilin	Note clicking behaviors retrieval	Successor SID tokens	SE.
	User gender lookup	{female, male, unknown}	ST.
	User age-group lookup	10 age groups, including unknown	ST.
	Device platform lookup	{Android, iOS, HarmonyOS}	ST.
JDSearch	Product interaction behavior	Successor SID tokens	SE.
KuaiSAR	Short video viewing behaviors (i.e., GSU) retrieval	Successor SID tokens	SE.

Table 10: Key training and inference hyperparameters.

Training		*AGR Fine-tuning*		*Inference*	
Optimizer	AdamW	Voting hit rate τ	0.8	Beam size K	$\{32, 50, 64\}$
Learning rate	1×10^{-4}	Loss clip $[\alpha_{\min}, \alpha_{\max}]$	$[0, 1]$	Tool quota M	1
Weight decay	0.01	Second stage epochs (Industry / Public)	$1/10$	Top-E (Attn)	5
Batch size	1024	Attention heads (Attn)	8		

Note: Model training and offline evaluation are conducted on a V100 GPU cluster, whereas the online A/B test is served on RTX 4090 GPUs.

as Qwen. As Figure 3 illustrates, its input and output spaces are asymmetric: the output contains generated SIDs and tool-call decisions, whereas the decoder input contains SIDs and externally returned observations. A generated SID is fed back autoregressively; a generated tool token instead triggers external execution, and the returned observation is fed into the next decoding step. This interface represents every tool call and observation with one decoding position and uses distinct positional encodings to separate observations from SID decisions. The quota M is maintained separately for every beam-decoding path. A quota of M calls therefore adds at most M decoding steps

to each path, so an L-token SID requires at most $L + M + 1$ steps including $\texttt{eos}$. Tool calls are neither fixed to predetermined positions nor required to alternate with SIDs. At each incomplete depth, the model may select either a valid SID successor or an available tool. The method described in Section 4.2 determines an admissible position at which to invoke each tool so as to maximize the discriminability of the feature. Theorem 3 further characterizes the position at which the feature is most informative w.r.t. sequential features.

Example 1 (*AGR* Workflow)**.** Figure 3 illustrates how *AGR* interleaves SID generation with feature acquisition for the query "Fighting Zombies". After the first SID token is generated, the prefix may lead to either the movie or game branch, making the next prediction uncertain. The model therefore emits the intent-tool token t_{I} and receives the game-intent observation o_{I} : Game, which steers decoding toward games while suppressing irrelevant movie candidates. Further along the game branch, query relevance alone cannot distinguish among candidates such as *Minecraft* and *Plants vs. Zombies*. The model thus emits the GSU-tool token t_{G} to acquire user preferences from historical behaviors. As shown in Figure 4, **Vote** retains items whose SIDs exactly match the decoded prefix (A, B, and D) and returns the most frequent successor SID, $\texttt{s3_7}$, indicating a preference for *Minecraft*. In contrast, **Attn** exploits the residual clustering structure of SIDs: summing the embeddings of the decoded SID tokens reconstructs the prefix representation at the current decoding depth. It then retrieves the Top-E historical behaviors whose item representations are closest to this reconstructed prefix representation, thereby including approximate matches such as C, and aggregates their successor SIDs via lightweight multi-head attention. Both implementations provide fine-grained preference signals that steer decoding toward the user's preferred *Minecraft* branch while suppressing irrelevant candidates.

Algorithm 1 Normal trie-based beam search.

Require: query q, user features u, beam size K, SID length L, SID trie Tr, generative retrieval model $\mathbb{P}_\theta$.
Ensure: top-K relevant items.

Beams $\mathcal{B} \leftarrow \{([\texttt{bos}], \emptyset, 0)\}$
for $j = 1$ **to** $L + 1$ **do**
  next layer candidates $\mathcal{C} \leftarrow \emptyset$
  for all $(\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z) \in \mathcal{B}$ **do**
    $\forall v,\ \ell(v) \leftarrow \log \mathbb{P}_\theta(v \mid q, u_{\text{vis}}, \mathbf{v}_{\text{in}}),\ \ \mathbf{s}_{\leqslant i} \leftarrow \mathbf{v}_{\text{out}} \setminus \{\texttt{eos}\}$
    for all $v \in \{\texttt{eos}\} \cup \{\text{next layer SIDs in Tr with prefix } \mathbf{s}_{\leqslant i}\}$ **do**
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\mathbf{v}_{\text{in}} \oplus v, \mathbf{v}_{\text{out}} \oplus v, z + \ell(v))\}$ ▷ $\mathbf{v}_{\text{out}} \oplus v$: append v to the end of list $\mathbf{v}_{\text{out}}$
  $\mathcal{B} \leftarrow \text{Top } K_{(\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z) \in \mathcal{C}}(z)$
return items mapped from $\{\mathbf{v}_{\text{out}} \setminus \{\texttt{eos}\} : (\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z) \in \mathcal{B}\}$

Algorithm 2 Tool-augmented trie-based beam search.

Require: query q, user features u, beam size K, SID length L, SID Trie Tr, tool set $\mathcal{T}$, tool quota M, *AGR* model $\mathbb{P}_\theta$.
Ensure: top-K relevant items.

Beams $\mathcal{B} \leftarrow \{([\texttt{bos}], \emptyset, 0)\}$, finish set $\mathcal{F} \leftarrow \emptyset$
for $j = 1$ **to** $L + M + 1$ **do**
  next layer candidates $\mathcal{C} \leftarrow \emptyset$
  for all $(\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z) \in \mathcal{B} \setminus \mathcal{F}$ **do**
    $\forall v,\ \ell(v) \leftarrow \log \mathbb{P}_\theta(v \mid q, u_{\text{vis}}, \mathbf{v}_{\text{in}})$
    for all $v \in \text{NEXT}(\text{Tr}, \mathbf{v}_{\text{out}})$ **do**
      if $v = t \in \mathcal{T}$ **then**
        $\mathbf{s}_{\leqslant i} \leftarrow \mathbf{v}_{\text{out}} \setminus (\mathcal{T} \cup \{\texttt{eos}\}),\ o_t \leftarrow f_t(q, u, \mathbf{s}_{\leqslant i}),\ (\mathbf{v}'_{\text{in}}, \mathbf{v}'_{\text{out}}) \leftarrow (\mathbf{v}_{\text{in}} \oplus o_t, \mathbf{v}_{\text{out}} \oplus t)$
      else $(\mathbf{v}'_{\text{in}}, \mathbf{v}'_{\text{out}}) \leftarrow (\mathbf{v}_{\text{in}} \oplus v, \mathbf{v}_{\text{out}} \oplus v)$ ▷ $\mathbf{v} \oplus a$: append a to the end of list $\mathbf{v}$
      $\mathcal{C} \leftarrow \mathcal{C} \cup \{(\mathbf{v}'_{\text{in}}, \mathbf{v}'_{\text{out}}, z + \ell(v))\}$
  $\mathcal{B} \leftarrow \text{Top } K_{(\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z) \in \mathcal{C} \cup \mathcal{F}}(z / |\mathbf{v}_{\text{out}}|)$
  $\mathcal{F} \leftarrow \{(\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z) \in \mathcal{B} : \mathbf{v}_{\text{out}} \text{ ends with } \texttt{eos}\}$
return items mapped from $\{\mathbf{v}_{\text{out}} \setminus (\mathcal{T} \cup \{\texttt{eos}\}) : (\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z) \in \mathcal{B}\}$
function NEXT(Tr, $\mathbf{v}_{\text{out}}$)
  $\mathbf{s}_{\leqslant i} \leftarrow \mathbf{v}_{\text{out}} \setminus (\mathcal{T} \cup \{\texttt{eos}\})$
  if $i = L$ **then return** $\{\texttt{eos}\}$
  $\mathcal{Y} \leftarrow \{\text{next-layer SIDs in Tr with prefix } \mathbf{s}_{\leqslant i}\}$
  if $|\mathbf{v}_{\text{out}} \cap \mathcal{T}| < M$ **then** $\mathcal{Y} \leftarrow \mathcal{Y} \cup \mathcal{T}$
  return $\mathcal{Y}$

Beam search. The **blue-gray** lines in Algorithm 2 highlight the core changes relative to the normal trie-based beam search in Algorithm 1. Both algorithms maintain candidates $(\mathbf{v}_{\text{in}}, \mathbf{v}_{\text{out}}, z)$ and expand trie-valid SID successors. Their main differences are: First, conventional decoding uses the fixed $L+1$ horizon (Algorithm 1, line 2), whereas *AGR* reserves M additional steps and exposes tool tokens only while quota remains (Algorithm 2, line 2 and line 18). Second, the conventional update appends every predicted token to both sequences (Algorithm 1, line 7); *AGR* instead appends a tool token only to $\mathbf{v}_{\text{out}}$ and its external observation only to $\mathbf{v}_{\text{in}}$, while retaining the conventional update for SIDs (Algorithm 2, lines 8). Third, conventional beam search ranks the current fixed-length expansions by z (Algorithm 1, line 8); *AGR* jointly compares active and completed variable-length paths using the length-normalized score $z/|\mathbf{v}_{\text{out}}|$ (Algorithm 2, lines 11). Finally, the conventional return removes only `eos` (Algorithm 1, line 9), whereas *AGR* additionally removes tool tokens to recover the same SID-only output space (Algorithm 2, line 13).

A.4 Data Construction and Training

Training-data construction. Section 4.2 presents the overall procedure for constructing tool-augmented supervision. When multiple tools are simultaneously applicable under a limited tool quota M, we prioritize tools with higher coverage in the training data. Eq. 8 and Eq. 9 use $\mathcal{X}$ and $\mathcal{D}_{\text{train}}$, respectively, to determine tool insertion positions: the former enables the model, while learning the empirical distribution from $\mathcal{D}_{\text{train}}$, to retrieve the true distribution of the required structural features along SID in $\mathcal{X}$ through tool calls, while the latter intends to inject less popular preferences through target items in $\mathcal{D}_{\text{train}}$. Example 2 provides a complete worked example illustrating the considerations for *structured feature tools*, including when each tool is applicable and where its observation should be inserted.

Example 2 (Tool-augmented Training Data Construction)**.** Follow the query "Fighting Zombies" in Example 1, Figure 6 shows the golden SID path "`s1_1` → `s2_1` → `s3_2` → ..." of the positive item. Each path contains 10 items. The numbers of items with the intent o_{I} : Game under `s3_1`, `s3_2`, `s3_3`, and `s3_4` are respectively 8, 9, 2, and 1. Therefore, along the golden path, the numbers of *Game* items are $20/40$ under `s1_1`, $17/20$ under `s2_1`, and $9/10$ under `s3_2`.

- **"When" is a tool applicable?** Suppose the tool returns $o_t = o_{\text{I}}$: Game. It can be invoked because the query intent matches the intent of the positive item and the intent is absent from the keyword "Fighting Zombies" and user features.
- **"Where" to insert the feature?** The concentrations along the golden path are $\rho_i(o_t) = \log(20/40)$ under `s1_1`, $\rho_{i+1}(o_t) = \log(17/20)$ under `s2_1`, and $\rho_{i+2}(o_t) = \log(9/10)$ under `s3_2`. The corresponding gains are $\log(17/20) - \log(20/40) = \log(1.7)$ and $\log(9/10) - \log(17/20) = \log(1.06)$. Because the first gain is larger, the tool is invoked after `s1_1`, before generating `s2_1`.

The example above illustrates how to construct tool-augmented training data. Each tool's insertion depth maximizes the increase of discriminability in Eq. 7. The selected tool token is inserted into the target output, while its observation appears only at the corresponding position in the shifted decoder input. During inference, each beam path independently decides at which decoding depth to invoke a tool, subject to its own per-path call quota.

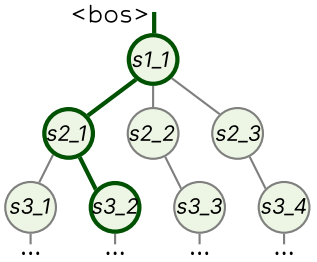


Figure 6: Training data construction example.

B Interpretive Analysis

B.1 An Interpretive View of Dynamic Feature Interaction

This section provides an interpretive account of dynamic feature interaction. It is not intended as a general performance guarantee for *AGR*. Instead, it explains a simple design intuition: **as the decoded SID prefix becomes more fine-grained, the information that is useful for predicting the next SID token may also change.**

Theorem 1 (Contextual Incremental Information)**.** A feature observation o_t is useful w.r.t. model $\mathbb{P}_\theta$ only if it adds information beyond the current context (i.e., q, u_{vis} and $\mathbf{s}_{\leqslant i}$).

Proof. Consider the conditional mutual information at the current context by definition:

$$I_\theta(s_{i+1}; o_t \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i}) = \mathbb{E}\left[\mathbb{D}_{\text{KL}}(\mathbb{P}_\theta(s_{i+1} \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i}, o_t) \,\|\, \mathbb{P}_\theta(s_{i+1} \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i}))\right]. \tag{12}$$

If this quantity is zero, the nonnegativity of the KL divergence implies that

$$\mathbb{P}_\theta(s_{i+1} \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i}, o_t) = \mathbb{P}_\theta(s_{i+1} \mid q, u_{\text{vis}}, \mathbf{s}_{\leqslant i}) \quad \text{almost surely.} \tag{13}$$

Hence, o_t provides no additional predictive information for the next SID token at the current context (i.e., $q, u_{\text{vis}}, \mathbf{s}_{\leqslant i}$). The contrapositive proves the theorem. □

Although conditional mutual information is nonnegative in expectation, the predictive gain of a realized observation can vary across contexts. This consideration is particularly important in industrial search and recommendation systems, where numerous features are available to characterize complex and evolving user preferences, yet generative retrieval models can incorporate only limited additional information during decoding because of efficiency and scalability constraints. Feature injection must therefore be selective, retaining observations that improve prediction at the current prefix while avoiding those that provide no gain or reduce the probability of the correct next SID token. Our method implements this context-dependent selection through tool invocation and observation injection (Section 4); next, we analyze this mechanism through experiments on tool selection and invocation.

Generative Retrieval Knapsack Problem (GR-KP). In *AGR*, the feature selection is implemented through tool interaction. Given the current context, predicting a tool token $t \in \mathcal{T}$ invokes its associated function f_t and returns an observation $o_t = f_t(q, u, \mathbf{s}_{\leqslant i})$. The returned observation token is then injected into the decoder to influence subsequence prediction. We formulate this feature injection as a *Generative Retrieval Knapsack Problem (GR-KP)*. At context q, u_{vis} and $\mathbf{s}_{\leqslant i}$, invoking tool t incurs cost $c_t > 0$ and yields context-dependent value $v_t(\mathbf{s}_{\leqslant i})$, which summarizes the contribution of its returned observation to decoding process. Let $\gamma_t \in [0, 1]$ denote the allocation assigned to each tool t. Under a finite model capacity C_θ, the feasible allocation set Γ and its value V are

$$\Gamma = \left\{ \boldsymbol{\gamma} = (\gamma_t)_{t\in\mathcal{T}} \,\middle|\, 0 \leqslant \gamma_t \leqslant 1,\ \sum_{t\in\mathcal{T}} c_t\gamma_t \leqslant C_\theta \right\}, \quad V(\mathbf{s}_{\leqslant i}, \boldsymbol{\gamma}) = \sum_{t\in\mathcal{T}} \gamma_t v_t(\mathbf{s}_{\leqslant i}). \tag{14}$$

Here, fractional allocation provides a compact estimator of the benefit of feature selection in a capacity-limited generative model. As discussed in Section 3, *static injection* assigns a single allocation $\boldsymbol{\gamma}$ shared across all prefixes, whereas *dynamic injection* assigns a separate allocation $\boldsymbol{\gamma}_i$ to the corresponding prefix $\mathbf{s}_{\leqslant i}$. Thus, the optimal expected cumulative utility under *static* and *dynamic injection* are

$$\mathcal{J}_{\text{static}} = \max_{\boldsymbol{\gamma}\in\Gamma} \mathbb{E}\left[\sum_i V(\mathbf{s}_{\leqslant i}, \boldsymbol{\gamma})\right], \qquad \mathcal{J}_{\text{dynamic}} = \mathbb{E}\left[\sum_i \max_{\boldsymbol{\gamma}_i\in\Gamma} V(\mathbf{s}_{\leqslant i}, \boldsymbol{\gamma}_i)\right]. \tag{15}$$

Theorem 2 (Dynamic Injection is Better)**.** When the usefulness of tool-derived features depends on the prefix $\mathbf{s}_{\leqslant i}$, dynamic injection is at least as effective as static injection, i.e., $\mathcal{J}_{\text{dynamic}} \geq \mathcal{J}_{\text{static}}$.

Proof. Recall that static injection selects one allocation $\boldsymbol{\gamma} \in \Gamma$ and uses it for every prefix: $\mathcal{J}_{\text{static}} = \max_{\boldsymbol{\gamma}\in\Gamma} \mathbb{E}[\sum_i V(\mathbf{s}_{\leqslant i}, \boldsymbol{\gamma})]$. A dynamic injection is allowed to choose an allocation $\boldsymbol{\gamma}_i(\mathbf{s}_{\leqslant i}) \in \Gamma$ after observing the current prefix. In particular, it can imitate any static policy by setting $\boldsymbol{\gamma}_i(\mathbf{s}_{\leqslant i}) = \boldsymbol{\gamma}$, for all $\mathbf{s}_{\leqslant i}$. Thus, every static allocation is feasible for the dynamic injection, and therefore

$$\mathcal{J}_{\text{dynamic}} = \mathbb{E}\left[\sum_i \max_{\boldsymbol{\gamma}_i\in\Gamma} V(\mathbf{s}_{\leqslant i}, \boldsymbol{\gamma}_i)\right] \geq \max_{\boldsymbol{\gamma}\in\Gamma} \mathbb{E}\left[\sum_i V(\mathbf{s}_{\leqslant i}, \boldsymbol{\gamma})\right] = \mathcal{J}_{\text{static}}. \tag{16}$$

□

This indicates that *dynamic injection* is a more dynamic and superior feature selection planning compared to *static injection*.

Theorem 3 (Equivalent Criteria for Sequential Feature Placement)**.** For a sequential feature, the position with the largest probability gain for the target next SID is equivalently the position that maximizes $\rho_{i+1}(o_{t,i}) - \rho_i(o_{t,i})$ and minimizes $\rho_i(o_{t,i})$.

Proof. Assume that the distribution learned by a sufficiently trained model approximates the empirical distribution of the training data: $\mathbb{P}_\theta \sim \mathbb{P}_{\mathcal{D}_{\text{train}}}$. Following Eq. 4, consider the logarithm of the probability-gain ratio r_i for predicting s^+_{i+1} after conditioning on the observation $o_{t,i}$ in addition to the prefix $\mathbf{s}^+_{\leqslant i}$:

$$\log r_i = \log \left[\frac{\mathbb{P}_\theta \left(s^+_{i+1} \mid \mathbf{s}^+_{\leqslant i}, o_{t,i} \right)}{\mathbb{P}_\theta \left(s^+_{i+1} \mid \mathbf{s}^+_{\leqslant i} \right)} \right] \approx \log \left[\frac{\mathbb{P}_{\mathcal{D}_{\text{train}}} \left(s^+_{i+1} \mid \mathbf{s}^+_{\leqslant i}, o_{t,i} \right)}{\mathbb{P}_{\mathcal{D}_{\text{train}}} \left(s^+_{i+1} \mid \mathbf{s}^+_{\leqslant i} \right)} \right]. \tag{17}$$

By Bayes' rule, we have

$$\mathbb{P}_{\mathcal{D}_{\text{train}}} \left(s^+_{i+1} \mid \mathbf{s}^+_{\leqslant i}, o_{t,i} \right) = \frac{\mathbb{P}_{\mathcal{D}_{\text{train}}} \left(o_{t,i} \mid \mathbf{s}^+_{\leqslant i}, s^+_{i+1} \right) \cdot \mathbb{P}_{\mathcal{D}_{\text{train}}} \left(s^+_{i+1} \mid \mathbf{s}^+_{\leqslant i} \right)}{\mathbb{P}_{\mathcal{D}_{\text{train}}} \left(o_{t,i} \mid \mathbf{s}^+_{\leqslant i} \right)}. \tag{18}$$

Substituting Eq. 18 into Eq. 17 and canceling the common term yields

$$\begin{aligned} \log r_i &\approx \log \mathbb{P}_{\mathcal{D}_{\text{train}}} \left(o_{t,i} \mid \mathbf{s}^+_{\leqslant i}, s^+_{i+1} \right) - \log \mathbb{P}_{\mathcal{D}_{\text{train}}} \left(o_{t,i} \mid \mathbf{s}^+_{\leqslant i} \right) \\ &= \log \mathbb{P}_{\mathcal{D}_{\text{train}}} \left(o_{t,i} \mid \mathbf{s}^+_{\leqslant i+1} \right) - \log \mathbb{P}_{\mathcal{D}_{\text{train}}} \left(o_{t,i} \mid \mathbf{s}^+_{\leqslant i} \right) \\ &= \rho_{i+1}(o_{t,i}) - \rho_i(o_{t,i}). \end{aligned} \tag{19}$$

Since the logarithm is strictly increasing, and the distributional approximation preserves the ordering across admissible positions,

$$\arg\max_i r_i \iff \arg\max_i \left[\rho_{i+1}(o_{t,i}) - \rho_i(o_{t,i}) \right]. \tag{20}$$

Moreover, for every correct sequential prediction satisfying Eq. 6, having $o_{t,i} = s^+_{i+1}$. Under the next-token prediction setting, given the prefix $\mathbf{s}^+_{\leqslant i+1}$, $\rho_{i+1}(o_{t,i})$ remains unchanged and can therefore be treated as a constant C', indicating that

$$\rho_{i+1}(o_{t,i}) - \rho_i(o_{t,i}) = C' - \rho_i(o_{t,i}), \tag{21}$$

and hence

$$\arg\max_i \left[\rho_{i+1}(o_{t,i}) - \rho_i(o_{t,i}) \right] \iff \arg\min_i \rho_i(o_{t,i}). \tag{22}$$

Therefore,

$$\arg\max_i r_i \iff \arg\max_i \left[\rho_{i+1}(o_{t,i}) - \rho_i(o_{t,i}) \right] \iff \arg\min_i \rho_i(o_{t,i}). \tag{23}$$

□

Takeaway. Theorem 1 shows that a tool is useful only when it provides information beyond the current prefix. Theorem 2 shows that adapting tool selection to each prefix is at least as effective as using a fixed selection. Theorem 3 shows that, for sequential features, the position with the largest probability gain can be identified by maximizing the change in ρ, equivalently by minimizing its current value. Together, these results motivate when, which, and where tools should be invoked during decoding.

C Further Ablation Study and Analysis

Observations contribute genuine feature information, not merely extra decoding steps. To verify that performance gains stem from the specific feature information carried by tool observations rather than from additional reasoning tokens, we replace per-observation tokens with a single shared learnable token, equivalent to a late-reasoning approach (*AGR* w/ shared observations). As shown in Table 11, this reasoning variant already outperforms the Baseline, confirming that extra decoding steps alone provide a non-trivial benefit. Yet *AGR* with distinct observation tokens achieves further gains, demonstrating that the observation tokens carry substantive feature information beyond what additional steps can provide.

Table 11: Observation analysis.

Method	Recall50
Baseline	0.2863
AGR w/ shared observation	0.2901
AGR	**0.3264**

Note: All tools are enabled in this ablation.

Structured feature tools outperform the direct encoding approach. In the *AGR* model, structured feature tools are incorporated into the decoding process through a token, depending on whether the relevant information is required at the current decoding step. An intuitive alternative is to directly integrate this information into the model's encoding to evaluate its potential benefits. As shown in Table 12, directly encoding the relevant information improves performance compared to the baseline (from 0.2863 to 0.3013). However, the encoding strategy employed by *AGR* achieves even better results, with a score of 0.3137. This demonstrates that introducing relevant information on demand during the decoding process reduces the inclusion of noisy information, thereby enabling more efficient information encoding.

Table 12: Structured injection vs. *AGR*.

Method	Recall@50
Baseline	0.2863
Static encoding for all structured features	0.3013
AGR w/ all structured feature tools only	**0.3137**

***AGR*'s gains persist across backbone architectures and model scales.** To examine the effects of backbone architecture and model scale, we evaluate UniSearch and *AGR* using BART (Lewis et al., 2020) and Qwen3 backbones with 0.6B, 1.7B, and 4B parameters (Yang et al., 2025). As shown in Table 13, Qwen3-0.6B outperforms the similarly sized BART backbone, suggesting that stronger pretrained language modeling provides benefits beyond parameter count alone. Scaling Qwen3 generally improves performance, with the 4B backbone achieving the best overall results. More importantly, *AGR* consistently outperforms UniSearch under every matched backbone, demonstrating that its improvements cannot be attributed solely to model capacity. These results confirm that backbone scaling and *AGR* provide complementary gains: larger pretrained models offer a stronger foundation, while explicit feature selection and targeted injection further improve SID generation across architectures and scales.

Table 13: Effect of backbone and model scaling on generative *AGR*.

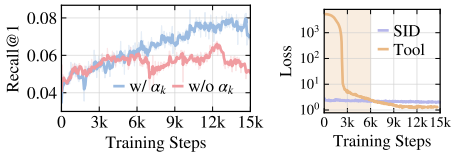
Method	Recall@50	MRR@50
BART		
UniSearch	0.3153	0.2563
AGR	**0.3264**	**0.2699**
Qwen3-0.6B		
UniSearch	0.3250	0.2772
AGR	**0.3347**	**0.2784**
Qwen3-1.7B		
UniSearch	0.3288	0.2843
AGR	**0.3350**	**0.2876**
Qwen3-4B		
UniSearch	0.3369	0.2960
AGR	**0.3444**	**0.2989**

The impact of fusion modes in sequential feature tools: attention-based fusion is most accurate; voting is the practical default. As discussed in Section 4.1, we implement two fusion modes for sequential feature tools (*Vote* and *Attn*) and introduce *Mean* as an additional baseline. *Mean* filters GSUs by prefix matching and averages their embeddings uniformly, discarding candidate importance variations; as expected, it yields the weakest results (Table 14). *Attn* achieves the best performance by relaxing strict prefix matching and learning adaptive fusion weights, confirming that candidate importance is both heterogeneous and learnable. Nevertheless, since *Vote* reaches comparable performance at substantially lower deployment cost, we adopt it as the default throughout this paper.

Table 14: GSU observation fusion strategy.

Observation fusion variant	Recall@50
Mean	0.3167
Vote	0.3237
Attn	0.3313

Note: only GSU tool is enable for this ablation.



(a) Tool-loss rescaling. (b) Loss scale.

Figure 7: Tool-loss rescaling and loss scale.

Tool loss rescaling prevents newly introduced tool tokens from disrupting SID semantics. Stage-II training introduces tool tokens on top of the SID generation ability learned in Stage-I, creating a scale mismatch between the pretrained SID objective and the newly added tool objective. To test whether this mismatch matters, we compare training with and without the batch-wise rescaling coefficient α_k in Eq. 10. As shown in Figure 7(a), rescaling consistently improves Recall@1, while Figure 7(b) shows a clear early-stage gap between SID- and tool-token loss magnitudes. This confirms that optimizing tool tokens at their raw scale can disturb the pretrained SID semantics, whereas α_k calibrates the two objectives and enables stable tool-enhanced semantic learning.

D Related Works

Generative retrieval and semantic identifiers. Distinct from traditional discriminative retrieval (Xie et al., 2026; Zheng et al., 2024), generative retrieval reformulates retrieval as conditional sequence generation over discrete identifiers (Chen et al., 2025c; Tay et al., 2022). Early work focused on document retrieval, where DSI stores query–document associations in model parameters (Tay et al., 2022), NCI introduces structured identifiers and prefix-aware decoding (Wang et al., 2022), and SEAL uses document-derived textual access paths (Bevilacqua et al., 2022). These studies established that the identifier space is learnable and that errors at different decoding positions have different retrieval consequences (Sun et al., 2023a; Zeng et al., 2024). Generative recommendation extends this idea to multi-token item semantic identifiers (SIDs). TIGER quantizes item representations with RQ-VAE and generates hierarchical SIDs (Rajput et al., 2023), making coarse-to-fine constrained decoding a standard formulation. Subsequent work shows that SID quality depends on semantic consistency, discrimination, codebook utilization, and tree balance (Ju et al., 2026), while autoregressive SID trees may also restrict ranking expressiveness (Hou et al., 2026). *AGR* builds on this SID-based generative retrieval paradigm, but shifts the focus from how identifiers are constructed to how external evidence should interact with SID generation.

Feature conditioning in generative retrieval. Real-world GR systems rely on heterogeneous auxiliary information, including user profiles, behavioral histories, scenario context, freshness signals, and intent features. Existing methods usually incorporate such information before decoding or through a fixed injection pattern (Wang et al., 2025). DiffusionGS performs query-conditioned denoising over candidate items (Li et al., 2026b); DualGR separates long- and short-term user interests (Yi et al., 2026); GLASS retrieves user histories conditioned on coarse SIDs (Cao et al., 2026); and OneReason partitions behavioral histories into temporal streams (Team et al., 2026). These methods show that request-aware evidence selection is important, and they move beyond globally encoding all context indiscriminately. However, their injection mechanisms are still largely predetermined: evidence is selected before generation, kept fixed after a coarse decision, or injected at predefined network locations. They do not formulate feature acquisition itself as an explicit decoding-time action. *AGR* differs by expanding the generative action space with tool tokens, allowing the model to decide whether a feature is needed, which feature to acquire, and where in the SID hierarchy the acquired evidence should affect subsequent generation.

Industrial GR systems and optimization. Industrial systems pursue GR for unification, cross-domain reuse, and serving efficiency. UniSearch jointly trains generators and encoders for video and live search (Chen et al., 2025c); OneSearch develops task-aware SID construction for e-commerce (Chen et al., 2025a); IntSR shares models across search and recommendation (Yan et al., 2025); OneRec unifies retrieval and ranking (Deng et al., 2025); and OneMall extends GR across multiple scenarios (Zhang et al., 2026b). Complementary work improves SID construction, collision control, and long-term maintenance (Wang et al., 2026c; Fu et al., 2026b; Baikalov et al., 2026; Fang et al., 2026), and generative interfaces have also been extended to reranking and query recommendation (Yang et al., 2026; Shao et al., 2025). Beyond maximum-likelihood training, preference-oriented methods align GR with ranking quality or business value: UniSearch uses ranking signals (Chen et al., 2025c), while related methods apply preference optimization to e-commerce (Li et al., 2024), adapt DPO to hierarchical SIDs (Chen et al., 2026b), use listwise optimization (Fu et al., 2026a), generative flow networks (Wang et al., 2026a), token-level signals (Lin et al., 2026),

iterative preference construction (Deng et al., 2025), page-wise supervision (Zou et al., 2026), and list-level value optimization (Xue et al., 2026). Efficiency methods accelerate decoding through draft verification (Wang et al., 2026b), process reward models (Guo et al., 2026), and load-aware computation (Xue et al., 2026). These efforts optimize different components of large-scale GR pipelines, whereas *AGR* targets a complementary bottleneck: avoiding noisy and expensive static feature injection by acquiring useful features on demand during decoding.

Reasoning and agentic retrieval. Recent work introduces intermediate reasoning to improve intent understanding in search and recommendation. OneSearch-V2 constructs keyword-based chains of thought (Chen et al., 2026a); OneRec-Think interleaves textual reasoning with generation (Liu et al., 2025); SCoTER transfers structured reasoning from LLMs (Jiang et al., 2025); DiffuReason combines latent thinking with diffusion refinement (Jiang et al., 2026); and OneReason organizes recommendation around perception and cognition (Team et al., 2026). Agentic recommendation further allows models to actively acquire external evidence. KP-Agent combines LLM agents with domain tools (Long et al., 2025), and ChainRec learns to route and terminate tool chains (Li et al., 2026a). However, general-purpose LLM planning typically introduces sequential inference, external orchestration, and tool-chain latency, which are difficult to accommodate in latency-constrained retrieval. Unlike these agentic recommenders, *AGR* treats feature access as a lightweight action inside native constrained SID decoding. Tool invocation is not a separate planning loop, but an interleaved part of beam search that directly reshapes the probability distribution over subsequent SID tokens.

E Case Study

Case overview. We present six representative requests to illustrate how *AGR* selectively invokes user- and query-dependent tools during generation: *World Cup Portugal* combines GSU, Freshness, and Intent; *Typhoon News* mainly uses Freshness; *Python Course* and *Study Application* use GSU to infer the user's learning and application goals; *Jeju Island* mainly uses Intent; and *Fireworks Festival* uses GSU to capture visual and aesthetic preferences. We first detail the tool-guided inference of *World Cup Portugal*; the other five cases are shown afterward.

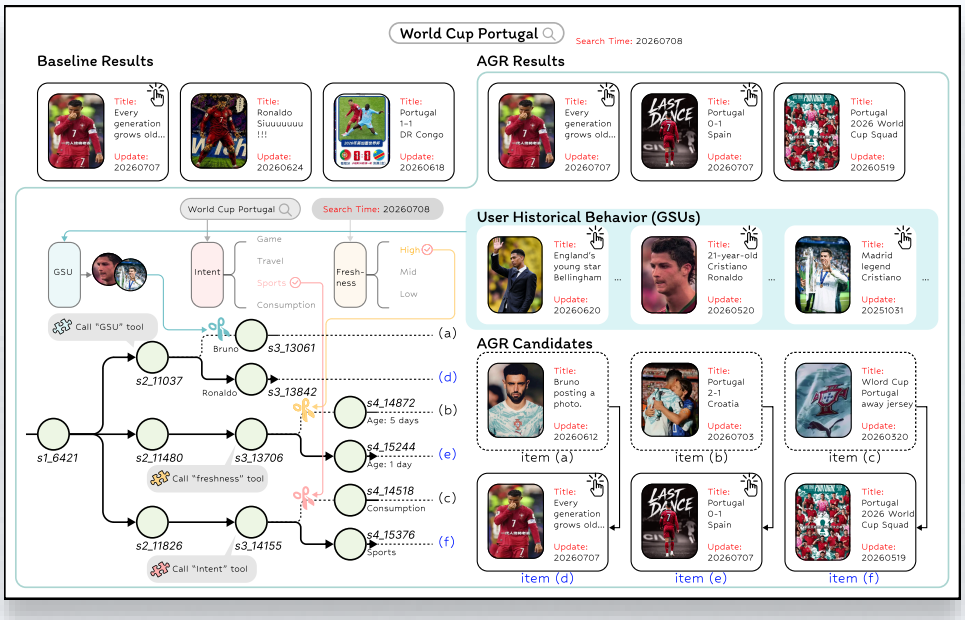


Figure 8: Case 1: Top-3 results for "World Cup Portugal"

Example 3 (Detailed inference for "World Cup Portugal")**.** The user had recently consumed several football videos and then issued the request "World Cup Portugal." The historical behaviors suggest an interest not only in the broad topic, but specifically in recent Portugal matches and Cristiano Ronaldo. The baseline captures the coarse topic, yet mixes a recent Ronaldo video with generic highlights and an older Portugal–DR Congo match (Figure 8). It therefore misses the request's personalized, temporal, and

intent-level requirements. *AGR* resolves these requirements through three tool-guided decoding paths. The showed calls below occur independently on three different beam paths:

- **Personalization through GSU.** At the prefix $[\texttt{s1_6421}, \texttt{s2_11037}]$, SID-only decoding favors $\texttt{s3_13061}$, leading to the Bruno Fernandes video (a). *AGR* instead invokes the GSU tool. Matching the prefix against the user's history yields five votes for $\texttt{s3_13842}$ versus one for $\texttt{s3_13061}$. The returned observation $o_{\mathrm{G}}: \texttt{s3_13842}$ redirects generation to the consumed Ronaldo video (d).
- **Temporal correction through Freshness.** At $[\texttt{s1_6421}, \texttt{s2_11480}, \texttt{s3_13706}]$, the original branch leads to an older July 3 match against Croatia (b). The Freshness tool classifies the query as high-freshness and returns $o_{\mathrm{F}}: \mathrm{High}$, shifting generation toward $\texttt{s4_15244}$. This retrieves the most recent report—Portugal's July 7 elimination by Spain (e).
- **Semantic disambiguation through Intent.** At $[\texttt{s1_6421}, \texttt{s2_11826}, \texttt{s3_14155}]$, the competing branch leads to a Portugal away-jersey video (c). The Intent tool recognizes that the user seeks Sports information and returns $o_{\mathrm{I}}: \mathrm{Sports}$. This suppresses merchandise-oriented content and redirects generation toward $\texttt{s4_15376}$, retrieving a video about Portugal's 2026 World Cup squad (f).

Together, these three independently tool-guided beam paths produce a top-3 ranked set that covers the user's personal preference, need for up-to-date results, and sports-intent content; the first two returned videos were subsequently clicked.

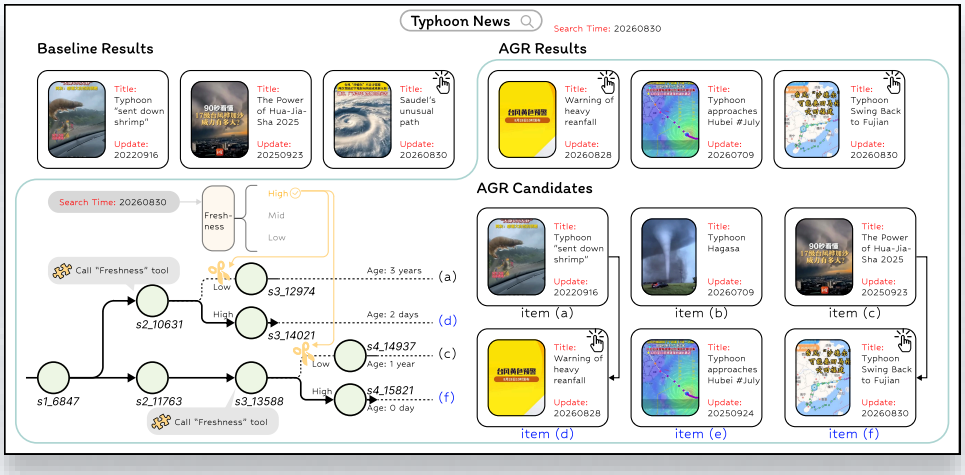


Figure 9: Case 2: Top-3 results for "Typhoon News"

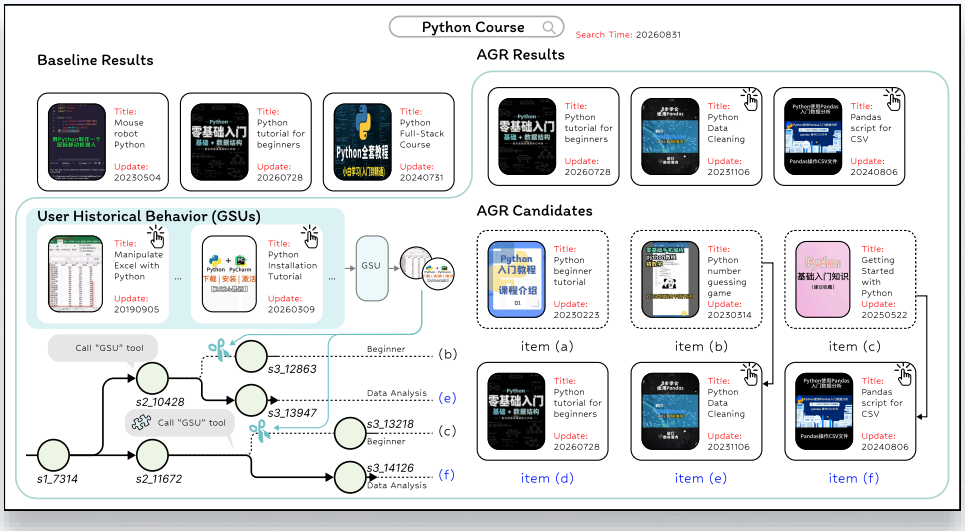


Figure 10: Case 3: Top-3 results for "Python Course"

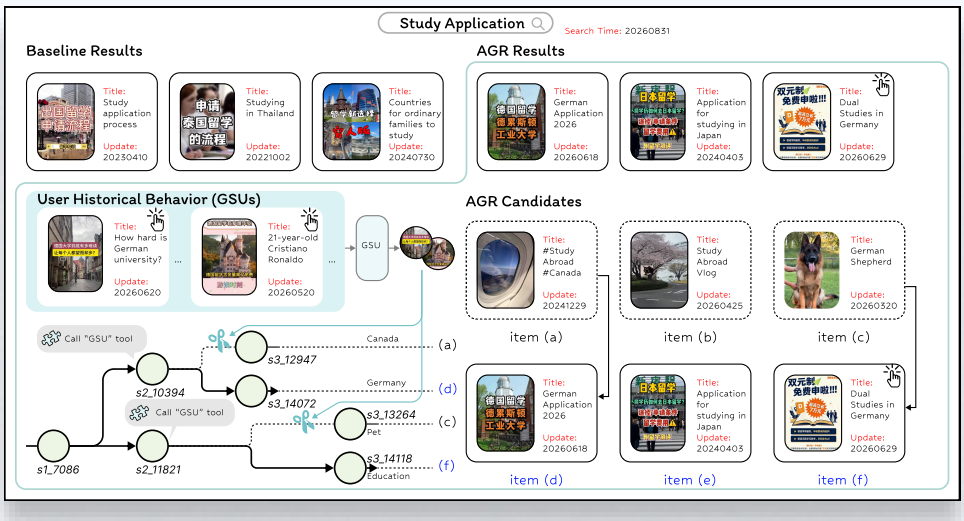


Figure 11: Case 4: Top-3 results for “Study Application”

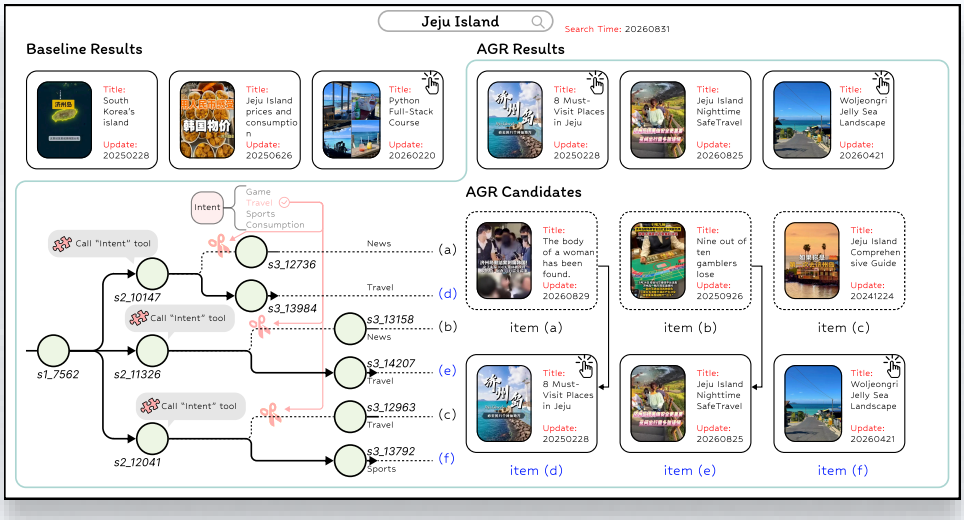


Figure 12: Case 5: Top-3 results for “Jeju Island”

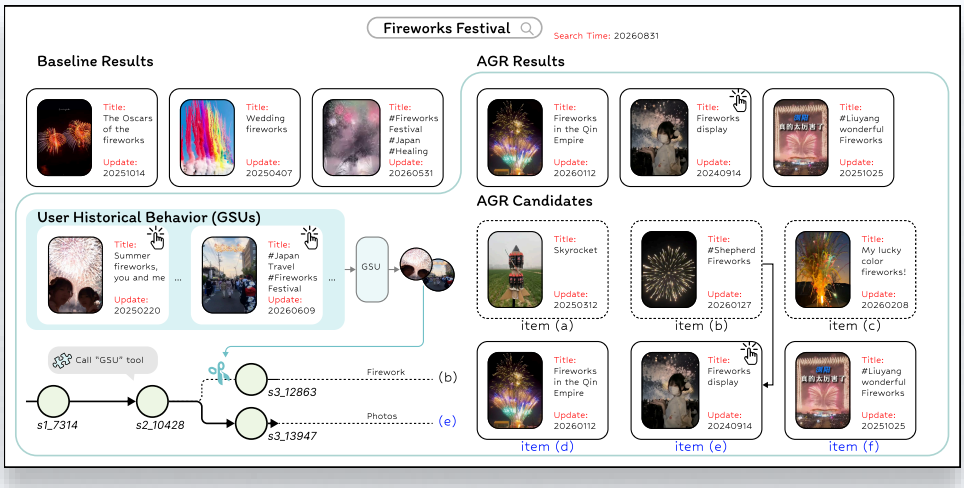


Figure 13: Case 6: Top-3 results for “Fireworks Festival”