IT TAKES TWO TO MATCH: CO-EVOLVING GENERATIVE RETRIEVER WITH REINFORCEMENT LEARNING

Anonymous authors
Paper under double-blind review

ABSTRACT

Recent work increasingly trains large language models (LLMs) for retrieval. Yet existing methods predominantly optimize generation solely on the query side, leaving item representations fixed. This one-sided adaptation restricts mutual alignment and caps overall retrieval performance. We introduce CoGR, a co-evolving reinforcement learning framework that jointly trains both query-side and item-side LLM generators. Each generator produces a compact set of discrete keywords for retrieval, preserving full compatibility with existing keyword-based infrastructure. CoGR first employs supervised fine-tuning to establish a warm-started, aligned keyword space. It then applies co-evolving reinforcement learning, alternately optimizing each generator against the opposite side's frozen index. Both sides optimize the same query-to-item F_1 objective: the query side receives F_1 directly, while the item side receives a counterfactual marginal reward designed to isolate each item's impact on retrieval performance. Evaluated against 10 representative sparse, dense, and generative retrieval baselines, CoGR achieves the strongest overall retrieval performance on both an internal APP Marketplace dataset and the public WANDS benchmark, improving F_1 over the strongest baseline by 10.9% and 36.1%, respectively. Further analysis demonstrates stable co-evolution dynamics and the emergence of more specific, balanced keyword representations.

1 INTRODUCTION

Retrieval is the first stage of modern search and advertising systems, selecting a candidate set from a large item universe for downstream ranking and auction. Because downstream stages operate only on the retrieved candidates, retrieval errors are largely irreversible: missed relevant items cannot be recovered later, while irrelevant items increase the burden on subsequent stages. An effective retriever must therefore balance broad coverage with a precise candidate set, reflecting the fundamental trade-off between recall and precision (Chowdhury, 2010).

Classical lexical retrieval relies on term matching over inverted indexes, typically scored with methods such as BM25 (Robertson & Zaragoza, 2009). Within this family, keyword-based matching remains prevalent in sponsored search, where advertisers bid directly on keywords associated with their ads. Dense retrieval instead maps queries and items into a shared continuous space for similarity-based matching (Karpukhin et al., 2020; Xiong et al., 2020). More recently, generative retrieval uses autoregressive generation of item identifiers (Tay et al., 2022; Wang et al., 2022; Zeng et al., 2024a), but often depends heavily on identifier design and faces challenges in decoding scalability and generalization (Li et al., 2025b).

Recent work has increasingly explored LLMs for retrieval, leveraging their strong capabilities in semantic understanding. Prompting-based approaches use LLMs for query rewriting, keyword expansion, and retrieval augmentation (Gao et al., 2023; Wang et al., 2023a; Ma et al., 2023; Liu et al., 2025), while more recent methods explicitly train LLMs using retrieval feedback (Jiang et al., 2025a; Li et al., 2025a; Yao et al., 2025). Despite these advances, LLM generation is still primarily optimized on one side of the retrieval process, most often the query side, and used to augment a downstream retriever. Such one-sided optimization can become a bottleneck, as the item-side representations cannot adapt to changes on the query side. This raises a natural question: *Can we jointly optimize representations for both queries and items and directly match them in the resulting space?*

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.

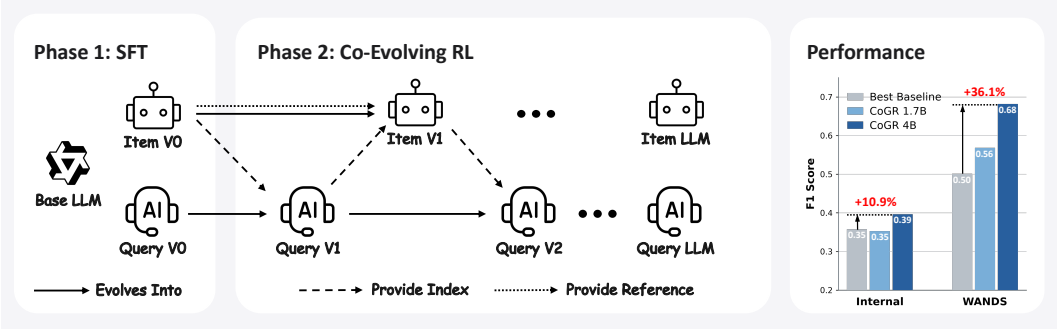


Figure 1: **Overall CoGR training pipeline and performance.** CoGR consists of an SFT stage followed by a co-evolving RL stage that alternately optimizes the item and query LLMs over multiple rounds. CoGR achieves strong performance gains on both the internal app marketplace and the public WANDS benchmark.

To this end, we propose CoGR, a retrieval framework that trains separate LLMs to generate keywords for queries and items. Given a query or an item, the corresponding generator produces a compact keyword set, and retrieval is performed directly by matching these sets through an inverted index. Moreover, this keyword-based formulation remains compatible with existing retrieval infrastructure and sponsored search business requirements.

The key challenge is to align the two generated keyword spaces so that semantically relevant query–item pairs can be reliably matched. CoGR addresses this challenge with the two-stage training pipeline shown in Figure 2, consisting of supervised fine-tuning (SFT) followed by co-evolving reinforcement learning (RL). The SFT stage primarily serves as a warm start for the subsequent RL stage by establishing an initially aligned keyword space. Specifically, we supervise the query-side generator using keywords from relevant items. Starting from this initialization, we alternately optimize the query- and item-side generators with GRPO (Shao et al., 2024). The query-side generator is directly rewarded by the retrieval F_1 achieved with its generated keywords. The item-side generator instead receives a counterfactual marginal reward, which measures the change in retrieval F_1 when an item's keyword set is replaced. During each update step, the opposite-side index is held fixed, providing a stable retrieval environment for optimization. Through this alternating procedure, the query- and item-side keyword spaces progressively co-evolve.

Empirically, we compare CoGR against 10 representative baselines spanning sparse, dense, and generative retrieval. On both the internal APP Marketplace dataset and the public WANDS product-search benchmark (Chen et al., 2022), CoGR achieves the highest overall F_1, outperforming the strongest baseline by 10.9% and 36.1%, respectively. Further analysis shows stable training dynamics throughout co-evolving RL, while the generated keyword spaces become progressively more specific and better aligned across training rounds.

2 METHOD

Let $\mathcal{Q}$ denote the query set and $\mathcal{I}$ a fixed item universe, where items may correspond to advertisements, products, or documents depending on the retrieval setting. For each query $q \in \mathcal{Q}$, let $\mathrm{rel}(q) \subseteq \mathcal{I}$ denote its set of relevant items. CoGR learns two separate keyword generators: a query-side generator G^Q and an item-side generator G^I. Given a query q and an item i, they produce keyword sets $S_q = G^Q(q)$ and $S_i = G^I(i)$, respectively. Here, each keyword is a possibly multi-word phrase, and overlap is defined by exact phrase matching. Retrieval is performed by keyword overlap, so an item is retrieved whenever its generated keyword set overlaps with that of the query. The resulting retrieved item set is $I_{\mathrm{ret}}(q) = \{i \in \mathcal{I} : S_q \cap S_i \neq \emptyset\}$. To rank the matched items, we treat the keyword sets as bags of words and score them using BM25.

The training procedure consists of two stages, as illustrated in Figure 2. First, supervised fine-tuning (SFT) initializes the query- and item-side generators. Then, co-evolving reinforcement learning alternately optimizes the two generators for retrieval quality.

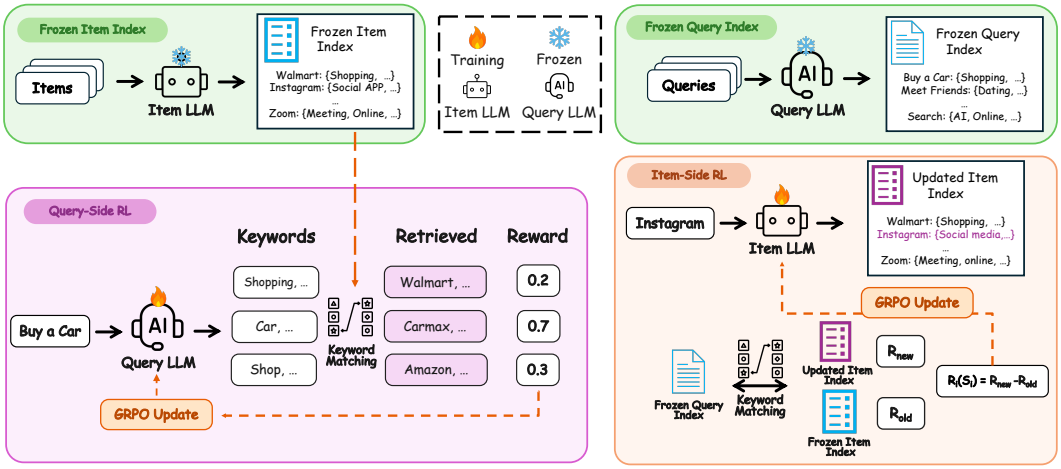


Figure 2: **Co-evolving reinforcement learning framework.** Training alternates between query-side and item-side optimization, with the opposite-side index kept frozen during each stage. *Left: Query-side RL.* The item LLM from the previous round constructs a frozen item index. For each query, the query LLM samples keyword sets, retrieves items through keyword matching, and is optimized with GRPO using the resulting retrieval F_1 as reward. *Right: Item-side RL.* The updated query LLM constructs a frozen query index. For each sampled keyword set S_i, we construct a counterfactual item index by replacing only the reference keywords of item i while leaving all other items unchanged. The item LLM is optimized with GRPO using the resulting marginal contribution to query-side retrieval quality, as defined in Equation 3.

2.1 Phase 1 — Initialization with Supervised Fine-tuning

Phase 1 of CoGR serves two purposes. First, it establishes an aligned keyword space between the query and item generators. Second, it provides RL with a meaningful initialization, ensuring sufficient initial recall to produce informative reward signals rather than requiring exploration from scratch.

Algorithm 1 SFT initialization of both sides.

Require: Query and item sets $\mathcal{Q}, \mathcal{I}$, base LLM G_0, budgets M, N
for all items $i \in \mathcal{I}$ **do**
  $S_i \leftarrow M$ keywords sampled from $G_0(i)$
end for
for all queries $q \in \mathcal{Q}$ **do**
  $\mathcal{B}_q \leftarrow \biguplus_{i \in \mathrm{rel}(q)} S_i$ ▷ multiset
  $S_q \leftarrow$ top-N most frequent keywords of $\mathcal{B}_q$
end for
$G^Q_{\mathrm{SFT}} \leftarrow \mathrm{SFT}(G_0; \{(q, S_q)\}_{q \in \mathcal{Q}})$
$G^I_{\mathrm{SFT}} \leftarrow \mathrm{SFT}(G_0; \{(i, S_i)\}_{i \in \mathcal{I}})$

The SFT data construction is summarized in Algorithm 1. We first use the original LLM to generate M item-side keywords for each item i, obtaining an initial keyword set S_i. We then construct query-side targets using the relevance labels. For each query q, we collect relevant items, pool their initial item-side keywords, and select the top-N most frequent keywords as the query-side target keyword set S_q. This construction ties each query-side keyword to keywords associated with its relevant items, thereby creating keyword overlap between relevant query–item pairs. Finally, we train the query-side generator G^Q on $\{(q, S_q)\}$ and the item-side generator G^I on $\{(i, S_i)\}$ with SFT, yielding the initialized policies G^Q_{SFT} and G^I_{SFT}.

2.2 Phase 2 — Co-Evolving Reinforcement Learning

Query-Side RL. After SFT, the RL stage further optimizes the two generators directly for retrieval quality. We adopt an alternating training paradigm: the query-side and item-side generators are updated in turn, while the inverted index constructed from the other side is kept frozen. This allows each generator to be optimized against a fixed retrieval environment rather than a moving target. In this section, we first introduce the specific reward design for each side, and then describe the alternating procedure that couples the two generators.

Given the item indexes generated by the frozen item-side LLM, query-side RL optimizes the query-side generator to improve retrieval quality. For each query $q \in \mathcal{Q}$, the query-side generator produces a keyword set S_q. The generated keywords are then matched against the item indexes to obtain the retrieved item set $I_{\text{ret}}(q)$. Ideally, the retrieved set should cover as many relevant items in $\text{rel}(q)$ while including as few irrelevant items as possible. We therefore use the F_1 score to evaluate retrieval quality:

$$P(q) = \frac{|\text{rel}(q) \cap I_{\text{ret}}(q)|}{|I_{\text{ret}}(q)|}, R(q) = \frac{|\text{rel}(q) \cap I_{\text{ret}}(q)|}{|\text{rel}(q)|}, F_1\big(I_{\text{ret}}(q), \text{rel}(q)\big) = \frac{2P(q)R(q)}{P(q)+R(q)}.^{1} \quad (1)$$

In industrial retrieval systems, both precision and recall are closely tied to business outcomes. Precision reflects the system's ability to filter out irrelevant items and therefore directly affects the user experience. Recall measures the coverage of relevant candidates passed to downstream ranking and bidding stages and is thus closely related to potential revenue. The F_1 score balances these two objectives and serves as a good reward signal.

To constrain the number of generated keywords, we further impose a maximum keyword budget $K_{\max}$. A sampled keyword set receives a reward of 0 if it exceeds the budget constraint:

$$\mathcal{R}_q(S_q) = \begin{cases} F_1\big(I_{\text{ret}}(q), \text{rel}(q)\big), & |S_q| \le K_{\max}, \\ 0, & |S_q| > K_{\max}. \end{cases} \quad (2)$$

Following the GRPO training paradigm (Shao et al., 2024), we sample multiple keyword sets for each query and compute a terminal reward for each rollout. The rewards are then normalized within each rollout group to obtain relative advantage signals for optimizing the query-side generator.

Item-Side RL. Item-side RL aims to improve query-to-item retrieval by refining item keywords. Given a candidate keyword set S_i for item i, we evaluate its quality by the change it induces in the overall retrieval quality. This requires isolating the contribution of S_i from those of all other items.

Specifically, at the beginning of each item-side update round, we freeze the query-side index and take the current item-side index as the reference state. We then construct a counterfactual index by replacing only the reference keywords of item i with S_i, while leaving the keywords of all other items unchanged. For each query q, let $I^{\text{ref}}_{\text{ret}}(q)$ and $I^{\text{cand}}_{\text{ret}}(q; S_i)$ denote the retrieved item sets under the reference and counterfactual indexes, respectively. We define the item-side reward as

$$\mathcal{R}_i(S_i) = \begin{cases} \underbrace{\sum_{q \in \mathcal{Q}} F_1\big(I^{\text{cand}}_{\text{ret}}(q; S_i), \text{rel}(q)\big)}_{R_{\text{new}}} - \underbrace{\sum_{q \in \mathcal{Q}} F_1\big(I^{\text{ref}}_{\text{ret}}(q), \text{rel}(q)\big)}_{R_{\text{old}}}, & \text{if } |S_i| \le K_{\max}, \\ -1, & \text{otherwise.} \end{cases} \quad (3)$$

The first term in equation 3 measures the aggregate retrieval quality under the counterfactual index, while the second measures that under the reference index. Their difference therefore isolates the effect attributable to S_i. This difference-based formulation also enables efficient reward computation, as queries whose F_1 scores remain unchanged cancel out in equation 3. Rather than running retrieval and computing F_1 for every query, we use the query-side inverted index together with cached query retrieval results to identify the affected queries and calculate only their contributions. We provide further implementation details in Appendix C.3.

Co-Evolving & Discussion. Phase 2 of CoGR is an iterative approach between query-side and item-side RL starting from the post-SFT models. The first query-side RL phase uses the G^I_{SFT} as index target. In later stages, each side is optimized against an index built by the latest model on the other side. This creates a co-evolution process in which the two generators progressively adapt to each other's keyword space and jointly evolve to improve the retrieval quality. The overall loop is summarized in Algorithm 2.

Algorithm 2 Co-evolving RL loop.

1: **Initialize:** Build item index $\mathcal{I}_0$ from G^I_{SFT}
2: **for** iteration $t = 1, 2, \ldots$ **do**
3: Train G^Q_t against frozen item index $\mathcal{I}_{t-1}$
4: Build query index $\mathcal{Q}_t$ from G^Q_t
5: Train G^I_t against frozen query index $\mathcal{Q}_t$
6: Build item index $\mathcal{I}_t$ from G^I_t
7: **end for**

[1] When the denominator of any term is zero, we define the corresponding term to be zero.

The framework also offers considerable flexibility. The same reward design can also be used with other RL algorithms such as PPO (Schulman et al., 2017). In applications where precision and recall have different business priorities, the standard F_1 reward can also be replaced by a weighted F-measure (Van Rijsbergen, 1979). Unless otherwise specified, we use the standard F_1 score throughout this work.

3 Experiment Setting

3.1 Datasets

We evaluate our method on two industrial search datasets. The first is an internal APP Marketplace search dataset consisting of de-identified, randomly sampled user queries. Each item corresponds to an application and is represented by its title and description. The second is WANDS (Chen et al., 2022), a public product-search dataset from Wayfair, where each item corresponds to a product and is likewise represented by its title and description. Both datasets provide categorical relevance annotations for $(\text{query}, \text{item})$ pairs, which we binarize into relevant and irrelevant classes (see Appendix B for details). We split the data along the query dimension while retaining the full item universe for both training and validation, so that performance on the validation set reflects generalization to unseen queries. Dataset statistics, including the number of relevant items per query, are summarized in Table 1. We focus on these datasets because they reflect the retrieval regime targeted in our application, where each query may correspond to a relatively large set of relevant items. In contrast, many conventional information retrieval datasets typically provide much sparser relevance annotations and thus less suited to our setting.

Table 1: Key statistics of Internal APP Marketplace search (Internal) and WANDS dataset.

Dataset	Queries (Train / Validation)	Item universe	Relevant items per query
Internal	13,500 / 1,500	39,600 Applications	$\approx 1,000$
WANDS	430 / 50	42,994 Products	≈ 200

3.2 Baseline Methods

We compare CoGR against three representative families of retrieval baselines, covering widely used methods from each category. Additional implementation details are provided in Appendix C.

- **Sparse (lexical) retrieval**, which scores query–item matches over sparse term representations. We include the classical BM25 (Robertson & Zaragoza, 2009) and the learned sparse retriever SPLADE-v2 (Formal et al., 2022).
- **Dense retrieval**, which maps queries and items into a shared embedding space and retrieves items based on vector similarity. We include DPR (Karpukhin et al., 2020) and ANCE (Xiong et al., 2020). To ensure a competitive comparison with modern dense baselines, we additionally evaluate Qwen3-Embedding-4B (Yang et al., 2025a), a strong 4B-scale embedding model comparable in model size to CoGR, under both zero-shot and ANCE-finetuned settings.
- **Generative retrieval**, which directly generates item identifiers using a sequence model. We include DSI (Tay et al., 2022), DSI-QG (Zhuang et al., 2022), and RIPOR (Zeng et al., 2024a). We further include DeepRetrieval (Jiang et al., 2025a), which trains a query-rewriting language model with reinforcement learning and is therefore closely related to our query-side RL formulation.

3.3 Training Setup

We instantiate two separate generators, one for the query side and one for the item side, using the same backbone architecture at each model scale: Qwen3-4B-Instruct for CoGR-4B and Qwen3-1.7B for CoGR-1.7B. Both Phase 1 and Phase 2 use the same prompt shown in Figure 5. We instruct the model to generate a comma-separated list of keywords directly, without chain-of-thought reasoning, to avoid additional inference overhead. In Phase 1 (SFT), we set the query-side top-N to 15 and the item-side keyword budget M to 10. In Phase 2 (co-evolving RL), we train both generators with GRPO (Shao et al., 2024) using the `verl` framework (Sheng et al., 2024), with the generated keyword set capped at $K_{\max} = 30$ on both sides. We alternate query- and item-side optimization for

Table 2: Retrieval performance on the validation sets. Columns suffixed with @100 (MRR@100, NDCG@100, P@100, R@100, F_1@100) are computed on the top-100 retrieved items, while P, R, and F_1 are the macro-averaged precision, recall, and F_1 over the full retrieved set. ❄ denotes that the Item-side parameters are frozen. The best value in each column within a dataset is shown in **bold**. Metrics at cutoffs 10 and 1000 are reported in Appendix D.

Dataset	Type	Method	MRR@100	NDCG@100	P@100	R@100	F_1@100	P	R	F_1
Internal	Sparse	BM25	0.6728	0.2654	0.3630	0.0377	0.0611	0.2453	0.1141	0.1056
		SPLADE-v2	0.5741	0.3365	0.4152	0.0588	0.0943	0.2706	0.4486	0.3019
	Dense	Qwen3-4B emb.	0.7255	0.4102	0.4436	0.0638	0.1022	0.2310	0.2689	0.2191
		DPR	0.5037	0.2860	0.3532	0.0504	0.0807	0.2408	0.4058	0.2700
		ANCE	0.6158	0.3866	0.4544	0.0636	0.1026	0.2671	0.4396	0.2971
		ANCE-Qwen4B	0.7572	0.4879	0.5691	0.0818	0.1312	0.3756	0.4354	0.3575
	Generative Retrieval	DSI	0.6323	0.3393	0.4234	0.0581	0.0935	0.2882	0.4790	0.3220
		DSI-QG	0.6644	0.3560	0.4413	0.0599	0.0966	0.2901	**0.4814**	0.3241
		RIPOR	0.6719	0.4146	0.4975	0.0674	0.1100	0.3371	0.3793	0.3167
		DeepRetrieval 4B	0.7454	0.4437	0.5128	0.0715	0.1159	0.2907	0.3333	0.2750
		CoGR❄ 1.7B	0.6450	0.3575	0.4310	0.0588	0.0958	0.3177	0.2289	0.2399
		CoGR 1.7B	0.7055	0.4456	0.5244	0.0739	0.1190	0.3523	0.4169	0.3527
		CoGR❄ 4B	0.7020	0.3958	0.4694	0.0661	0.1066	0.3335	0.2459	0.2617
		CoGR 4B	**0.7667**	**0.4930**	**0.5844**	**0.0844**	**0.1349**	**0.3976**	0.4569	**0.3963**
WANDS	Sparse	BM25	0.8575	0.7047	0.6508	0.4092	0.4263	0.4768	0.6149	0.4418
		SPLADE-v2	0.9090	0.7872	0.7340	**0.4506**	0.4678	0.5914	0.6050	0.4903
	Dense	Qwen3-4B emb.	0.6471	0.4718	0.4270	0.2572	0.2599	0.3371	0.3408	0.2634
		DPR	0.8861	0.7259	0.6794	0.4071	0.4222	0.5612	0.5581	0.4520
		ANCE	0.8903	0.7333	0.6932	0.4000	0.4218	0.5731	0.5485	0.4554
		ANCE-Qwen4B	0.9085	**0.7955**	0.7506	0.4498	0.4717	0.6136	0.6092	0.5012
	Generative Retrieval	DSI	0.5862	0.4706	0.4620	0.2396	0.2527	0.3994	0.3468	0.2890
		DSI-QG	0.8517	0.6594	0.6270	0.3659	0.3867	0.5051	0.5044	0.4023
		RIPOR	0.7905	0.5966	0.5704	0.3044	0.3283	0.4760	0.4205	0.3595
		DeepRetrieval 4B	0.8592	0.7029	0.6473	0.4072	0.4234	0.4802	0.6155	0.4431
		CoGR❄ 1.7B	0.8332	0.4652	0.5945	0.2123	0.2686	0.5073	0.3751	0.3664
		CoGR 1.7B	0.8903	0.6701	0.8398	0.3339	0.4236	0.7325	0.5503	0.5685
		CoGR❄ 4B	0.8111	0.5710	0.6494	0.2941	0.3554	0.5691	0.4696	0.4662
		CoGR 4B	**0.9225**	0.7146	**0.8417**	0.4059	**0.4778**	**0.7940**	**0.6885**	**0.6819**

five rounds, training the query generator for 10 GRPO epochs and the item generator for 5 epochs per round. We report the final checkpoint after five training rounds and do not perform validation-based checkpoint selection. Unless otherwise specified, subsequent analyses use CoGR-4B. Additional training and implementation details are provided in Appendix C.

4 Experimental Results

4.1 Main Results

Main results are reported in Table 2, including precision, recall, F_1, MRR, NDCG, and top-100 metrics, with additional results at cutoffs of 10 and 1000 provided in Appendix D. For CoGR, retrieval rankings are obtained using BM25 over the generated keywords, as described in Section 2. For baselines that output ranked lists, we select the retrieval cutoff that maximizes training F_1 and apply it unchanged to the validation set. Baseline checkpoints are selected by validation F_1, whereas CoGR uses the final checkpoint after five co-evolving rounds.

The key observations from Table 2 are as follows:

- CoGR achieves the strongest overall retrieval performance on both datasets, reaching F_1 scores of 0.396 and 0.682 on Internal and WANDS, respectively. Compared with the strongest baseline, ANCE-Qwen4B, this corresponds to relative improvements of 10.9% and 36.1%.
- Co-evolving the query and item representations is critical to this performance. CoGR substantially outperforms both CoGR❄and DeepRetrieval, which optimize the query side while keeping the item-side representations fixed. These results highlight the benefit of jointly adapting both sides to learn better-aligned representations for retrieval.

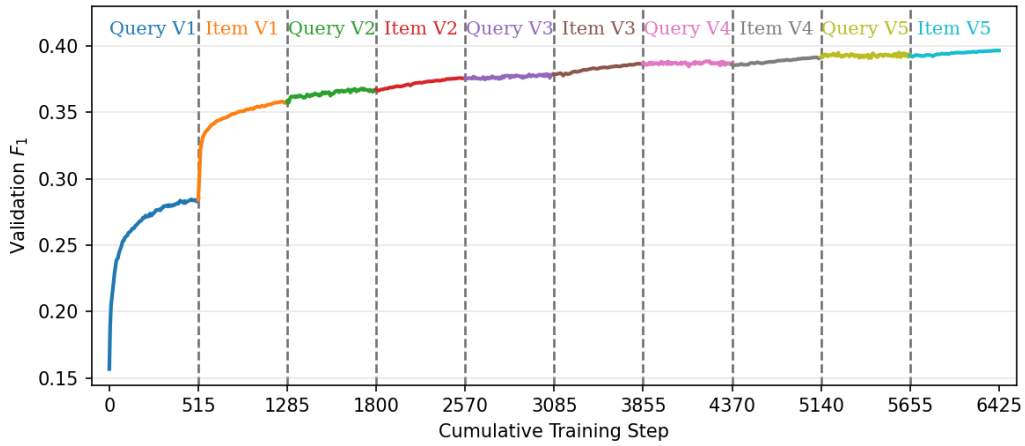


Figure 3: Validation F_1 over cumulative training steps on the Internal dataset. Starting from the SFT-initialized query and item generators, training proceeds for five co-evolving rounds ($V1$–$V5$), where $V1$ denotes the first query/item RL update pair after SFT initialization. Performance improves most sharply in $V1$ and continues to increase more gradually in later rounds.

4.2 Co-evolving Dynamics

Figure 3 traces the validation F_1 throughout the alternating RL process, starting from the SFT-initialized generators. Over five rounds of co-evolution, validation F_1 increases from approximately 0.16 to nearly 0.40, with the largest gains occurring in the first round. Subsequent query- and item-side updates provide smaller but consistent improvements, while largely preserving gains from the previous phase. This stable progression suggests that alternating co-evolution enables the two generators to progressively adapt to each other's evolving keyword representations.

4.3 Ablation on Training Design

We ablate three design choices in CoGR: the marginal item-side reward in Equation 3, the use of separate query- and item-side generators, and the SFT initialization in Phase 1. Specifically, *Transposed F_1* replaces the marginal item-side reward with a symmetric item-centric retrieval objective. Analogous to the query-side reward in Equation 2, we treat each item as a query and measure how well it retrieves its relevant queries:

$$\mathcal{R}_i^{\text{trans}}(S_i) = \begin{cases} F_1\big(Q_{\text{ret}}(i), \text{rel}(i)\big), & |S_i| \le K_{\max}, \\ 0, & |S_i| > K_{\max}, \end{cases}$$

where $Q_{\text{ret}}(i)$ denotes the set of queries retrieved for item i, and $\text{rel}(i)$ denotes the set of queries relevant to i. *Shared generator* uses a single generator for both query- and item-side keyword generation, so the co-evolving procedure alternately updates the same checkpoint for the two roles while maintaining separate reference policies. *No SFT* removes the Phase 1 initialization and starts co-evolving RL directly from the base Qwen3-4B checkpoint.

As shown in Table 3, all three variants using Qwen3-4B as the base model underperform the full CoGR model, supporting the effectiveness of each design choice. At the same time, all variants achieve reasonable retrieval performance, suggesting that the overall co-evolving framework remains robust under alternative reward designs, generator configurations, and initialization strategies.

Table 3: Ablation of training design choices on the Internal dataset.

Variant	P	R	F_1
Transposed F_1	0.3482	0.4462	0.3743
Shared generator	0.3678	0.4635	0.3798
No SFT	0.3800	0.4283	0.3751
CoGR (full)	0.3976	0.4569	**0.3963**

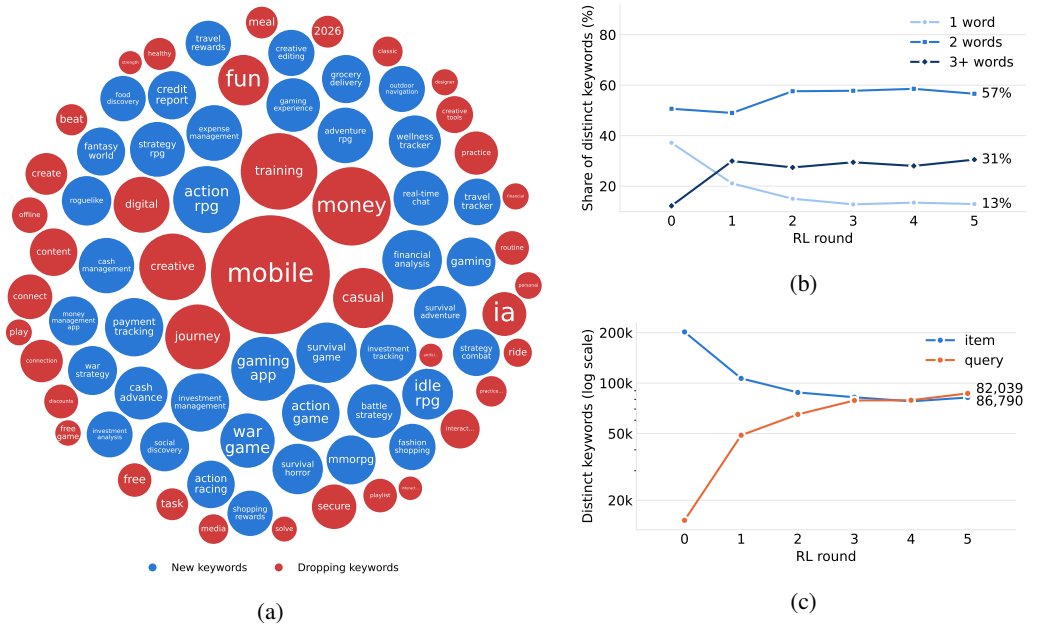


Figure 4: Keyword vocabulary dynamics under co-evolving RL on the Internal dataset. *(a)* Keywords added (blue) and dropped (red) between the post-SFT vocabulary and that after five RL rounds; bubble area is proportional to the number of entities associated with each keyword. *(b)* Distribution of distinct keywords by the number of words they contain across training rounds. For panels (a) and (b), the keyword vocabulary at each stage is defined as the union of all distinct query-side and item-side keywords. *(c)* Number of distinct indexed n-grams on the item (app) and query sides across rounds, shown on a log scale.

4.4 Analysis of Keyword Evolution

We analyze how the generated keyword space evolves throughout co-evolving RL by comparing the post-SFT generators with those obtained over five rounds of alternating optimization, as summarized in Figure 4.

A clear trend is that the learned keywords become increasingly specific. Co-evolving RL reduces the prevalence of generic unigrams while increasing the use of more descriptive multi-word keywords. In Figure 4a, keywords removed during training are often broad terms such as "mobile" and "fun", whereas newly introduced keywords tend to be longer and more specific, such as "war game" and "real-time chat". Figure 4b shows the same trend quantitatively: the proportion of unigrams decreases from 37% to 13%, while the proportion of keywords containing three or more words increases from 12% to 31%.

We further observe that the query- and item-side keyword spaces become more balanced over training. As shown in Figure 4c, the item-side vocabulary contracts substantially during the early RL rounds, whereas the query-side vocabulary expands. Their vocabulary sizes gradually approach each other and become comparable after approximately three to four rounds. Together, these results suggest that co-evolving RL not only encourages more specific keyword representations, but also progressively brings the two sides into a more balanced representational regime.

4.5 Ablation on Additional Information

We further examine how retrieval performance changes with the information provided to the query- and item-side generators. By default, the item-side input includes both the application title and description. We consider two variants: removing the item description, or augmenting the query-side prompt with search results from the existing search system as an additional hint.

Table 4 shows that additional contextual information can substantially improve retrieval quality. Removing item descriptions reduces F_1 from 0.3963 to 0.3759, indicating that descriptions provide useful semantic cues when application titles alone are insufficient. In contrast, augmenting the query-side input with search results improves both precision and recall, increasing F_1 to 0.4379. As illustrated by the qualitative examples in Table 8 in the Appendix, additional context helps resolve underspecified or misleading surface forms on both sides. On the query side, search results provide useful hints for ambiguous, misspelled, entity-centric, and non-English queries. On the item side, descriptions disambiguate opaque or ambiguous application titles and expose semantic information that is not available from the title alone. Overall, these results show that CoGR can naturally incorporate additional contextual information to further improve retrieval.

Table 4: Ablation on the information available to each generator. − description leaves the item side with the title only, and + search results additionally places search results in the query-side prompt.

Variant	P	R	F_1
CoGR (default)	0.3976	0.4569	**0.3963**
− Description	0.3875	0.4204	0.3759
+ Search Results	0.4381	0.5002	0.4379

5 Related Work

We focus here on LLM-based retrieval methods, including LLM-driven data synthesis and retrieval augmentation, representation construction, and retrieval-feedback optimization. Classical sparse/dense retrieval and generative retrieval, which primarily serve as baseline families in our experiments, are reviewed in Appendix A.

Prompting-based methods improve retrieval in several forms. Some directly rewrite queries or generate keyword-based query expansions (Ma et al., 2023; Li et al., 2024b; Liu et al., 2025). Others perform query expansion based on generated pseudo-documents or retrieved corpus evidence (Gao et al., 2023; Wang et al., 2023a; Mackie et al., 2023; Shen et al., 2024; Lei et al., 2024). Beyond query-side augmentation, recent work applies LLMs to both query and document sides for dual-side expansion or representation construction (Zhuang et al., 2024; Yang et al., 2025b).

More closely related to our setting, recent work explicitly trains LLMs using retrieval feedback. Early work trains query rewriters using feedback from retrieval or downstream generation (Ma et al., 2023; Wang et al., 2025; Nguyen et al., 2025). DeepRetrieval directly optimizes an LLM query generator with reinforcement learning using retrieval metrics as rewards (Jiang et al., 2025a). Subsequent work extends this paradigm toward reasoning-enhanced, retriever-specific, and process-level optimization (Qin et al., 2025; Zhu et al., 2025; Li et al., 2026; Ouyang et al., 2025; Ye et al., 2025). Other approaches jointly adapt query expansion and retrievers (Li et al., 2025a; Yao et al., 2025), or train multi-step search agents using retrieval and downstream task feedback (Jiang et al., 2025b; Jin et al., 2025).

CoGR jointly trains query- and item-side keyword generators whose outputs directly form the retrieval index. The two sides are alternately optimized under a shared objective, enabling co-adaptive lexical representations. This is also related in spirit to self-evolving LLM systems (Huang et al., 2026b;a).

6 Conclusion

We introduced CoGR, a co-evolving generative retrieval framework that trains LLMs to generate keywords for both query and item sides through alternating RL optimization. Experiments on internal APP Marketplace dataset and WANDS show consistent improvements over strong retrieval baselines. Further analysis reveals stable co-evolving dynamics and increasingly specific keyword representations throughout training.

This framework leaves several directions for future work. The reward could incorporate objectives beyond relevance, including downstream business metrics such as irrelevant-ad rate and revenue. In addition, our current system uses BM25 to rank the generated keyword representations. Learning a stronger ranking component, potentially jointly with the generators, may further improve retrieval quality.

AI USE STATEMENT

Generative AI tools were used to assist with language editing and LaTeX template migration. The authors reviewed all AI-assisted work and verified the resulting text, equations, references, and formatting. The authors take responsibility for the final content of this work, including all text, claims, and artifacts produced with the aid of generative AI.

REFERENCES

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.

Yan Chen, Shujian Liu, Zheng Liu, Weiyi Sun, Linas Baltrunas, and Benjamin Schroeder. Wands: Dataset for product search relevance assessment. In *European Conference on Information Retrieval*, pp. 128–141. Springer, 2022.

Gobinda G Chowdhury. *Introduction to modern information retrieval*. Facet publishing, 2010.

Zhuyun Dai and Jamie Callan. Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval*, pp. 1533–1536, 2020.

Nicola De Cao, Gautier Izacard, Sebastian Riedel, and Fabio Petroni. Autoregressive entity retrieval. *arXiv preprint arXiv:2010.00904*, 2020.

Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. Splade: Sparse lexical and expansion model for first stage ranking. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pp. 2288–2292, 2021.

Thibault Formal, Carlos Lassance, Benjamin Piwowarski, and Stéphane Clinchant. From distillation to hard negative sampling: Making sparse neural ir models more effective. In *Proceedings of the 45th international ACM SIGIR conference on research and development in information retrieval*, pp. 2353–2359, 2022.

Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. Precise zero-shot dense retrieval without relevance labels. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 1762–1777, 2023.

Chengsong Huang, Haolin Liu, Tong Zheng, Runpeng Dai, Langlin Huang, Jinyuan Li, Zongxia Li, Zhepei Wei, Yu Meng, and Jiaxin Huang. G-zero: Self-play for open-ended generation from zero data. *arXiv preprint arXiv:2605.09959*, 2026a.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026b.

Pengcheng Jiang, Jiacheng Lin, Lang Cao, Runchu Tian, SeongKu Kang, Zifeng Wang, Jimeng Sun, and Jiawei Han. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. *arXiv preprint arXiv:2503.00223*, 2025a.

Pengcheng Jiang, Xueqiang Xu, Jiacheng Lin, Jinfeng Xiao, Zifeng Wang, Jimeng Sun, and Jiawei Han. s3: You don't need that much data to train a search agent via rl. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21610–21628, 2025b.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *arXiv preprint arXiv:2503.09516*, 2025.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 conference on empirical methods in natural language processing (EMNLP)*, pp. 6769–6781, 2020.

Sunkyung Lee, Minjin Choi, and Jongwuk Lee. Glen: Generative retrieval via lexical index learning. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 7693–7704, 2023.

Yibin Lei, Yu Cao, Tianyi Zhou, Tao Shen, and Andrew Yates. Corpus-steered query expansion with large language models. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 393–401, 2024.

Alan Li, Daniel Cheng, Phillip Keung, Jungo Kasai, and Noah A Smith. Summarization-based document ids for generative retrieval with language models. In *Proceedings of the First Workshop on Advancing Natural Language Processing for Wikipedia*, pp. 126–135, 2024a.

Chaofan Li, Jianlyu Chen, Yingxia Shao, Chaozhuo Li, Quanqing Xu, Defu Lian, and Zheng Liu. Reinforced ir: A self-boosting framework for domain-adapted information retrieval. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22061–22073, 2025a.

Chenyi Li, Xinhui Tu, and Zaixiang Wang. Dvcqr: Dual-view conversational query rewriting with stage-wise reinforcement learning. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 22993–23014, 2026.

Minghan Li, Honglei Zhuang, Kai Hui, Zhen Qin, Jimmy Lin, Rolf Jagerman, Xuanhui Wang, and Michael Bendersky. Can query expansion improve generalization of strong cross-encoder rankers? In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2321–2326, 2024b.

Xiaoxi Li, Jiajie Jin, Yujia Zhou, Yuyao Zhang, Peitian Zhang, Yutao Zhu, and Zhicheng Dou. From matching to generation: A survey on generative information retrieval. *ACM Transactions on Information Systems*, 43(3):1–62, 2025b.

Yongqi Li, Nan Yang, Liang Wang, Furu Wei, and Wenjie Li. Multiview identifiers enhanced generative retrieval. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 6636–6648, 2023.

Yongqi Li, Nan Yang, Liang Wang, Furu Wei, and Wenjie Li. Learning to rank in generative retrieval. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 8716–8723, 2024c.

Jimmy Lin and Xueguang Ma. A few brief notes on deepimpact, coil, and a conceptual framework for information retrieval techniques. *arXiv preprint arXiv:2106.14807*, 2021.

Tongtong Liu, Zhaohui Wang, Meiyue Qin, Zenghui Lu, Xudong Chen, Yuekui Yang, and Peng Shu. Real-time ad retrieval via llm-generative commercial intention for sponsored search advertising. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 28936–28948, 2025.

Zheng Liu, Wei Zhang, Yan Chen, Weiyi Sun, Tianchuan Du, and Benjamin Schroeder. Towards generalizeable semantic product search by text similarity pre-training on search click logs. In *Proceedings of the Fifth Workshop on e-Commerce and NLP (ECNLP 5)*, pp. 224–233, 2022.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting in retrieval-augmented large language models. In *Proceedings of the 2023 conference on empirical methods in natural language processing*, pp. 5303–5315, 2023.

Iain Mackie, Shubham Chatterjee, and Jeffrey Dalton. Generative relevance feedback with large language models. In *Proceedings of the 46th international ACM SIGIR conference on research and development in information retrieval*, pp. 2026–2031, 2023.

Antonio Mallia, Omar Khattab, Torsten Suel, and Nicola Tonellotto. Learning passage impacts for inverted indexes. In *Proceedings of the 44th international ACM SIGIR conference on research and development in information retrieval*, pp. 1723–1727, 2021.

Kidist Amde Mekonnen, Yubao Tang, and Maarten de Rijke. Lightweight and direct document relevance optimization for generative information retrieval. In *Proceedings of the 48th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 1327–1338, 2025.

Duy A Nguyen, Rishi Kesav Mohan, Shimeng Yang, Pritom Saha Akash, and Kevin Chen-Chuan Chang. Minielm: A lightweight and adaptive query rewriting framework for e-commerce search optimization. In *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 6952–6964, 2025.

Jianmo Ni, Chen Qu, Jing Lu, Zhuyun Dai, Gustavo Hernandez Abrego, Ji Ma, Vincent Zhao, Yi Luan, Keith Hall, Ming-Wei Chang, et al. Large dual encoders are generalizable retrievers. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pp. 9844–9855, 2022.

Rodrigo Nogueira, Jimmy Lin, and AI Epistemic. From doc2query to doctttttquery. *Online preprint*, 6(2):1–3, 2019.

Yichen Ouyang, Lu Wang, Fangkai Yang, Pu Zhao, Chenghua Huang, Jianfeng Liu, Bochen Pang, Yaming Yang, Yuefeng Zhan, Hao Sun, et al. Token-level proximal policy optimization for query generation. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 31184–31198, 2025.

Xubo Qin, Jun Bai, Jiaqi Li, Zixia Jia, and Zilong Zheng. Reinforced query reasoners for reasoning-intensive retrieval tasks. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21261–21274, 2025.

Yingqi Qu, Yuchen Ding, Jing Liu, Kai Liu, Ruiyang Ren, Wayne Xin Zhao, Daxiang Dong, Hua Wu, and Haifeng Wang. Rocketqa: An optimized training approach to dense passage retrieval for open-domain question answering. In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 5835–5847, 2021.

Stephen Robertson and Hugo Zaragoza. The probabilistic relevance framework: Bm25 and beyond. *Foundations and trends® in information retrieval*, 4(1-2):1–174, 2009.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Tao Shen, Guodong Long, Xiubo Geng, Chongyang Tao, Yibin Lei, Tianyi Zhou, Michael Blumenstein, and Daxin Jiang. Retrieval-augmented retrieval: Large language models are strong zero-shot retriever. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 15933–15946, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

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, 2023.

Yubao Tang, Ruqing Zhang, Jiafeng Guo, Maarten de Rijke, Wei Chen, and Xueqi Cheng. Generative retrieval meets multi-graded relevance. *Advances in neural information processing systems*, 37:72790–72817, 2024.

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.

C Van Rijsbergen. Information retrieval: theory and practice. In *Proceedings of the joint IBM/University of Newcastle upon tyne seminar on data base systems*, volume 79, pp. 1–14. Butterworth-Heinemann Oxford, UK, 1979.

Liang Wang, Nan Yang, and Furu Wei. Query2doc: Query expansion with large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 9414–9423, 2023a.

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.

Yujing Wang, Hainan Zhang, Liang Pang, Binghui Guo, Hongwei Zheng, and Zhiming Zheng. Maferw: Query rewriting with multi-aspect feedbacks for retrieval-augmented large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 25434–25442, 2025.

Zihan Wang, Yujia Zhou, Yiteng Tu, and Zhicheng Dou. Novo: Learnable and interpretable document identifiers for model-based ir. In *Proceedings of the 32nd ACM International Conference on Information and Knowledge Management*, pp. 2656–2665, 2023b.

Lee Xiong, Chenyan Xiong, Ye Li, Kwok-Fung Tang, Jialin Liu, Paul Bennett, Junaid Ahmed, and Arnold Overwijk. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*, 2020.

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*, 2025a.

Tianchi Yang, Minghui Song, Zihan Zhang, Haizhen Huang, Weiwei Deng, Feng Sun, and Qi Zhang. Auto search indexer for end-to-end document retrieval. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pp. 6955–6970, 2023.

Yu Yang, Feng Tian, and Ping Chen. Tcde: Topic-centric dual expansion of queries and documents with large language models for information retrieval. *arXiv preprint arXiv:2512.17164*, 2025b.

Sijia Yao, Pengcheng Huang, Zhenghao Liu, Yu Gu, Yukun Yan, Shi Yu, and Ge Yu. Expandr: teaching dense retrievers beyond queries with llm guidance. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 19047–19065, 2025.

Xiaopeng Ye, Chen Xu, Chaoliang Zhang, Zhaocheng Du, Jun Xu, Gang Wang, and Zhenhua Dong. Q-prm: Adaptive query rewriting for retrieval-augmented generation via step-level process supervision. In *EMNLP (Findings)*, pp. 15113–15128, 2025.

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, 2024a.

Hansi Zeng, Chen Luo, and Hamed Zamani. Planning ahead in generative retrieval: Guiding autoregressive generation through simultaneous decoding. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 469–480, 2024b.

Hailin Zhang, Yujing Wang, Qi Chen, Ruiheng Chang, Ting Zhang, Ziming Miao, Yingyan Hou, Yang Ding, Xupeng Miao, Haonan Wang, et al. Model-enhanced vector index. *Advances in Neural Information Processing Systems*, 36:54903–54917, 2023.

Yujia Zhou, Zhicheng Dou, and Ji-Rong Wen. Enhancing generative retrieval with reinforcement learning from relevance feedback. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 12481–12490, 2023.

Yujia Zhou, Jing Yao, Zhicheng Dou, Yiteng Tu, Ledell Wu, Tat-Seng Chua, and Ji-Rong Wen. Roger: Ranking-oriented generative retrieval. *ACM Transactions on Information Systems*, 42(6): 1–25, 2024.

Changtai Zhu, Siyin Wang, Ruijun Feng, Kai Song, and Xipeng Qiu. Convsearch-r1: Enhancing query reformulation for conversational search with reasoning via reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 26558–26575, 2025.

Shengyao Zhuang, Houxing Ren, Linjun Shou, Jian Pei, Ming Gong, Guido Zuccon, and Daxin Jiang. Bridging the gap between indexing and retrieval for differentiable search index with query generation. *arXiv preprint arXiv:2206.10128*, 2022.

Shengyao Zhuang, Xueguang Ma, Bevan Koopman, Jimmy Lin, and Guido Zuccon. Promptreps: Prompting large language models to generate dense and sparse representations for zero-shot document retrieval. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pp. 4375–4391, 2024.

A Additional related work

Sparse and Dense Retrieval. Classical sparse retrieval uses lexical matching, such as TF–IDF and BM25 (Robertson & Zaragoza, 2009), while learned sparse methods retain the inverted-index interface but learn term weights or lexical expansions, including docTTTTTquery (Nogueira et al., 2019), DeepCT/DeepImpact (Dai & Callan, 2020; Mallia et al., 2021), uniCOIL (Lin & Ma, 2021), and SPLADE (Formal et al., 2021; 2022). Dense retrieval follows the two-tower paradigm, with representative methods including DPR (Karpukhin et al., 2020), ANCE (Xiong et al., 2020) with ANN-mined hard negatives, RocketQA (Qu et al., 2021) with improved negative sampling and denoising, and GTR (Ni et al., 2022) with scaled pretrained encoders. Sparse and dense retrieval remain widely used paradigms in large-scale retrieval systems.

Generative Retrieval. Generative retrieval replaces nearest-neighbor search with autoregressive generation of document identifiers. One line uses *numeric identifiers*. DSI (Tay et al., 2022; Zhuang et al., 2022) assigns documents semantic IDs and trains a sequence-to-sequence model to generate them from queries; NCI (Wang et al., 2022) improves this framework with query augmentation and prefix-aware decoding. Later methods learn or optimize identifiers rather than fixing them beforehand: GENRET (Sun et al., 2023) jointly learns document tokenization and retrieval, ASI (Yang et al., 2023) and MEVI (Zhang et al., 2023) explore learned document indexing, while RIPOR (Zeng et al., 2024a) constructs identifiers from retrieval-oriented quantized representations and optimizes relevance across identifier prefixes.

A parallel line uses *lexical identifiers* in the pretrained vocabulary space. GENRE (De Cao et al., 2020) uses document titles, SEAL (Bevilacqua et al., 2022) uses document substrings with constrained generation, and MINDER (Li et al., 2023) combines multiple lexical views. More recent work moves beyond fixed lexical identifiers: GLEN (Lee et al., 2023) and NOVO (Wang et al., 2023b) learn lexical identifiers from retrieval supervision, while ACID (Li et al., 2024a) constructs abstractive identifiers such as generated keyphrases or summaries. Orthogonal to identifier design, prior work also improves generative retrieval through ranking- or relevance-aligned objectives (Li et al., 2024c; Zhou et al., 2023; Mekonnen et al., 2025; Tang et al., 2024), knowledge distillation (Zhou et al., 2024), and decoding strategies such as PAG (Zeng et al., 2024b).

B Relevance Label Details

Both datasets provide categorical relevance labels for query–item pairs, which we convert into binary labels for constructing the gold set. For the Internal dataset, we label query–item pairs using an internal LLM-as-a-judge with a five-level scale: *excellent*, *good*, *acceptable*, *poor*, and *bad*. We treat pairs rated *acceptable* or better as relevant and the remaining pairs as irrelevant. For WANDS (Chen et al., 2022), human annotators provide three-level labels: *Exact*, *Partial*, and *Irrelevant*. WANDS constructs a high-coverage relevance pool through iterative mining and cross-referencing of potentially relevant products. Following prior evaluation practice on WANDS (Liu et al., 2022), we treat *Exact* and *Partial* pairs as relevant and all other query–item pairs as irrelevant.

Query-side generator G^Q

System: You expand user search queries into matching keywords.

User: List keywords (synonyms, related terms, item categories) that the query should match against. Output only a comma-separated list of keywords, no explanation.

User query: {query}

Item-side generator G^I

System: You extract search keywords from item titles and descriptions.

User: List keywords for lexical search matching. Output only a comma-separated list of keywords, no explanation.

Item title: {item}

Item Description: {description}

Figure 5: Prompts used to fine-tune the query-side (G^Q) and item-side (G^I) generators.

C Training Details

C.1 Prompts

The prompts used in phase 1 and 2 training are shown in Figure 5.

C.2 Hyperparameters

We use the default sampling hyperparameters recommended for Qwen for Phase 1 Item keyword initialization (Section 2.1), as well as for model evaluation and index construction in Phase 2 (Section 2.2). During RL training, we instead use unconstrained stochastic sampling to encourage exploration. The decoding hyperparameters for each setting are summarized in Table 5. For GRPO training, we use 8 NVIDIA B200 GPUs. The key hyperparameters for query- and item-side optimization are summarized in Table 6. We adopt a fully online RL setup, generating fresh rollouts from the current policy at each update to avoid off-policy training.

Table 5: Decoding hyperparameters used for RL rollouts, initialization, evaluation, and index construction.

Setting	Temp.	top-p	top-k	min-p
RL rollout	1.0	1.0	-1	0
Initialization / Evaluation / Indexing	0.7	0.8	20	0

Table 6: GRPO hyperparameters for item-side and query-side training. Hyperparameter names follow the original `verl` configuration fields.

Hyperparameter	Item side	Query side
Advantage estimator	GRPO	GRPO
`optim.lr`	10^{-6}	10^{-6}
`rollout.n`	8	8
`train_batch_size`	256	512
`ppo_mini_batch_size`	256	512
`ppo_micro_batch_size_per_gpu`	32	64
`n_gpus_per_node`	8	8
`max_response_length`	512	512
`use_kl_loss`	False	False
Epochs	5	10

C.3 Efficient Implementation of Item-Side Reward

Naively computing the item-side reward in Eq. 3 would require rebuilding the counterfactual item index and rerunning retrieval over all queries for every sampled rollout. We avoid this cost by exploiting the fact that each counterfactual rollout modifies the representation of only a single item.

At the beginning of each item-side training round, we cache the retrieval statistics under the reference item index. For each query q, we store the number of retrieved items, true positives, and relevant items:

$$n_q^{\text{ret}} = \left|I_{\text{ret}}^{\text{ref}}(q)\right|, \qquad n_q^{\text{tp}} = \left|I_{\text{ret}}^{\text{ref}}(q) \cap \text{rel}(q)\right|, \qquad n_q^{\text{rel}} = \left|\text{rel}(q)\right|.$$

The corresponding reference F_1 score is $F_1^{\text{ref}}(q) = \frac{2n_q^{\text{tp}}}{n_q^{\text{ret}}+n_q^{\text{rel}}}$.

For each item i, we additionally cache the set of queries that retrieve it under the reference item index $\mathcal{Q}_i^{\text{ref}} = \left\{q \in \mathcal{Q} : i \in I_{\text{ret}}^{\text{ref}}(q)\right\}$. Given a sampled keyword set S_i, we perform a single lookup against the frozen query-side inverted index to obtain the queries that would retrieve the candidate representation:

$$\mathcal{Q}_i^{\text{cand}} = \left\{q \in \mathcal{Q} : i \in I_{\text{ret}}^{\text{cand}}(q; S_i)\right\}.$$

Since the counterfactual index differs from the reference index only in the representation of item i, only queries whose retrieval status for i changes can contribute to the reward. The affected query set is therefore $\mathcal{Q}_i^{\Delta} = \mathcal{Q}_i^{\text{cand}} \triangle \mathcal{Q}_i^{\text{ref}}$, where $\triangle$ denotes the symmetric difference. For each affected query, the candidate representation either adds or removes item i from the retrieved set. Its candidate F_1 score can therefore be updated directly from the cached statistics:

$$F_1^{\text{cand}}(q) = \begin{cases} \dfrac{2\left(n_q^{\text{tp}} + \mathbb{I}[i \in \text{rel}(q)]\right)}{n_q^{\text{ret}} + 1 + n_q^{\text{rel}}}, & q \in \mathcal{Q}_i^{\text{cand}} \setminus \mathcal{Q}_i^{\text{ref}}, \\ \dfrac{2\left(n_q^{\text{tp}} - \mathbb{I}[i \in \text{rel}(q)]\right)}{n_q^{\text{ret}} - 1 + n_q^{\text{rel}}}, & q \in \mathcal{Q}_i^{\text{ref}} \setminus \mathcal{Q}_i^{\text{cand}}. \end{cases}$$

All unaffected queries have identical retrieved sets under the reference and counterfactual indexes, and hence identical F_1 scores. Their contributions therefore cancel in the reference-subtracted objective in Eq. 3. The item-side reward can thus be computed using only the affected queries:

$$\mathcal{R}_i(S_i) = \begin{cases} \displaystyle\sum_{q \in \mathcal{Q}_i^{\Delta}} \left(F_1^{\text{cand}}(q) - F_1^{\text{ref}}(q)\right), & |S_i| \le K_{\max}, \\ -1, & |S_i| > K_{\max}. \end{cases}$$

Thus, each valid item-side rollout requires only one lookup against the frozen query-side index, followed by constant-time F_1 updates for the affected queries. This avoids both rebuilding the full counterfactual item index and rerunning retrieval over the entire query set for every rollout.

D Retrieval Metrics at Additional Cutoffs

Overall, CoGR performs strongly on set-level retrieval metrics such as precision, recall, and F_1, while also achieving competitive or best MRR, indicating that relevant items are both well covered and surfaced early. For CoGR, when fewer items are retrieved than the evaluation cutoff, we append randomly sampled items from the remaining item universe to reach the cutoff. Its gains are less consistent on NDCG, which places greater emphasis on the relative ordering of highly ranked items. This is expected because CoGR currently uses a simple BM25 scorer over the generated keywords for ranking, without explicitly optimizing the ranking stage. Incorporating a stronger learned ranker is therefore a natural direction for future work.

Table 7: Validation-set query-side retrieval metrics at cutoffs 10 and 1000, using the same method grouping as Table 2. ❄ denotes that the Item-side parameters are frozen. The best value in each column within a dataset is shown in **bold**.

Dataset	Type	Method	MRR@10	NDCG@10	P@10	R@10	F_1@10	MRR@1000	NDCG@1000	P@1000	R@1000	F_1@1000
Internal	Sparse	BM25	0.6682	0.3281	0.4815	0.0066	0.0128	0.6729	0.1640	0.2453	0.1141	0.1056
		SPLADE-v2	0.5683	0.2660	0.4471	0.0069	0.0133	0.5744	0.3938	0.3054	0.3523	0.2895
	Dense	Qwen3-4B emb.	0.7212	0.4185	0.5653	0.0091	0.0176	0.7256	0.3339	0.2310	0.2689	0.2191
		DPR	0.4955	0.2250	0.3793	0.0057	0.0110	0.5041	0.3466	0.2677	0.3132	0.2551
		ANCE	0.6109	0.3348	0.5156	0.0079	0.0153	0.6161	0.4023	0.3062	0.3490	0.2888
		ANCE-Qwen4B	0.7544	**0.4261**	0.6445	0.0104	0.0201	0.7573	**0.4989**	0.3756	**0.4354**	0.3575
	Generative Retrieval	DSI	0.6265	0.2838	0.4846	0.0069	0.0134	0.6325	0.4113	0.3201	0.3738	0.3053
		DSI-QG	0.6597	0.3020	0.5121	0.0073	0.0141	0.6647	0.4172	0.3234	0.3753	0.3077
		RIPOR	0.6671	0.3460	0.5527	0.0079	0.0155	0.6722	0.4370	0.3371	0.3793	0.3167
		DeepRetrieval 4B	0.7422	0.4044	0.6154	0.0097	0.0187	0.7456	0.3992	0.2907	0.3333	0.2750
		CoGR❄ 1.7B	0.6402	0.3111	0.5046	0.0076	0.0147	0.6452	0.2747	0.3251	0.2068	0.2302
		CoGR 1.7B	0.7019	0.3888	0.5966	0.0092	0.0177	0.7057	0.4146	0.3806	0.3408	0.3296
		CoGR❄ 4B	0.6990	0.3578	0.5695	0.0090	0.0173	0.7022	0.2985	0.3431	0.2232	0.2517
		CoGR 4B	**0.7643**	0.4198	**0.6576**	**0.0106**	**0.0202**	**0.7668**	0.4621	**0.4269**	0.3810	**0.3716**
WANDS	Sparse	BM25	0.8556	0.7314	0.7816	0.0564	0.1006	0.8575	0.7907	0.3149	0.8330	0.3806
		SPLADE-v2	0.9090	**0.7861**	**0.8780**	0.0636	0.1125	0.9091	0.8530	0.2732	0.8818	0.3384
	Dense	Qwen3-4B emb.	0.6442	0.5129	0.5780	0.0456	0.0800	0.6476	0.5478	0.1759	0.5915	0.2150
		DPR	0.8850	0.7539	0.8460	0.0631	0.1113	0.8861	0.8199	0.2692	0.8662	0.3318
		ANCE	0.8900	0.7617	0.8580	0.0640	0.1125	0.8905	0.8133	0.2726	0.8467	0.3334
		ANCE-Qwen4B	0.9067	0.7816	0.8620	0.0603	0.1072	0.9085	**0.8591**	0.2785	**0.8883**	0.3447
	Generative Retrieval	DSI	0.5814	0.4717	0.5580	0.0343	0.0606	0.5867	0.5782	0.2244	0.6451	0.2632
		DSI-QG	0.8517	0.6827	0.7800	0.0548	0.0972	0.8521	0.7474	0.2544	0.7891	0.3067
		RIPOR	0.7902	0.6476	0.7500	0.0514	0.0907	0.7907	0.6857	0.2416	0.7259	0.2889
		DeepRetrieval 4B	0.8550	0.7320	0.7857	0.0570	0.1016	0.8592	0.7956	0.3280	0.8354	0.3915
		CoGR❄ 1.7B	0.8315	0.6089	0.7236	0.0468	0.0841	0.8333	0.4293	0.5336	0.3713	0.3726
		CoGR 1.7B	0.8903	0.7427	0.8391	0.0589	0.1050	0.8903	0.6188	0.7787	0.5515	0.5814
		CoGR❄ 4B	0.8079	0.6462	0.7408	0.0527	0.0941	0.8111	0.5184	0.5848	0.4736	0.4732
		CoGR 4B	**0.9209**	0.7537	0.8655	**0.0695**	**0.1162**	**0.9225**	0.7075	**0.8115**	0.6633	**0.6703**

E EXAMPLE GENERATED KEYWORDS

Table 8 provides qualitative examples of the keywords generated after Round 5, and examines how the outputs change when additional contextual information is provided. On the query side, we augment the original query with online search hints (Table 8); on the item side, we augment the application title with its description (Table 8). These examples illustrate how additional context can help resolve ambiguous or underspecified inputs and produce more semantically relevant keywords.

Table 8: Effect of additional context on the keywords generated after Round 5. Query-side generation is augmented with online search hints, while item-side generation is augmented with the application description.

(a) Query-side keywords with and without online search hints.

Query	w/o Search	w/ Search
family search	album, anniversary, birthday, birthdays, digital, photo, search, sharing, tracker, ...	genealogy, family history, genealogy app, family calendar, ancestry, family tree, ...
解压软件	relaxation, mindfulness, meditation, relaxation app, stress relief, puzzle game, mental health, ...	archive, compress, compression, extract, extractor, file, files, manager, rar, unzip, zip, ...
zenless	mindfulness, meditation, relaxation, meditation app, mental health, mindfulness app, stress relief, ...	action, adventure, battle, combat, defense, fantasy, royale, rpg, ...
duch bros	action, multiplayer, action game, arcade game, adventure game, action shooter, arcade, action rpg, ...	burger, coffee, deals, delivery, drink, fast, food, local, order, pizza, ...

(b) Item-side keywords with and without application descriptions.

Application	w/o Desc.	w Desc.
10000000	10000000, 1000	rpg, dungeon crawler, puzzle, action rpg, roguelike, adventure,. . .
Compound Quest	crypto, decentralized finance, dapp, yield farming, blockchain	compound words, word puzzles, vocabulary building, spelling skills, reading skills, esl learning, picture match
Ultra Plunge Tracker	dive, dive log, diving, swim	cold plunge, sauna, steam room, ice bath, heart rate, recovery, wellness, ai coach, hyperbaric chamber
Life is Slow	life is slow, slow life	nature, simulation, nature simulation, puzzle, relaxation, nature theme, adventure, nature game, nature therapy

F BASELINE METHODS

We compare against three families of retrieval baselines, with the base model used for each method summarized in Table 9. For all trainable baselines, we follow their standard training setups and use the same query- and item-side information available to CoGR whenever applicable, including the same item title and description fields and aligned prompting inputs. This ensures that performance differences are not driven by additional textual information or task-specific prompt engineering.

Table 9: Base models used for baselines.

Baseline	Base model
BM25	—
DPR	`bert-base-multilingual-uncased`
ANCE	`roberta-base`
SPLADE-v2	`naver/splade_v2_max`
Qwen3-4B emb.	`Qwen3-Embedding-4B`
ANCE-Qwen4B	`Qwen3-Embedding-4B`
DSI	`mT5-base`
DSI-QG	`mT5-base`
RIPOR	`mT5-base`
DeepRetrieval	`Qwen3-4B-Instruct`

For unconstrained precision, recall, and F_1, we tune the retrieval cutoff on the training split by selecting the value that maximizes training F_1, and then apply this fixed cutoff to the validation set. Checkpoints are selected according to their validation F_1

under the same protocol. Unless otherwise specified, trained baselines use 8 GPUs with distributed data parallelism and otherwise follow their standard optimization configurations.