RankEvolve: Automating the Discovery of Retrieval Algorithms via LLM-Driven Evolution

Jinming Nian
Santa Clara University
USA
jnian@scu.edu

Fangchen Li
Independent Researcher
USA
fangchen.li@outlook.com

Dae Hoon Park*
melten.ai
USA
dae.hoon.park@melten.ai

Yi Fang
Santa Clara University
USA
yfang@scu.edu

Abstract

Retrieval algorithms like BM25 and query likelihood with Dirichlet smoothing remain strong and efficient first-stage rankers, yet improvements have mostly relied on parameter tuning and human intuition. We investigate whether a large language model, guided by an evaluator and evolutionary search, can automatically discover improved lexical retrieval algorithms. We introduce RankEvolve[2], a program evolution setup based on AlphaEvolve, in which candidate ranking algorithms are represented as executable code and iteratively mutated, recombined, and selected based on retrieval performance across 12 IR datasets from BEIR and BRIGHT. RankEvolve starts from two seed programs: BM25 and query likelihood with Dirichlet smoothing. The evolved algorithms are novel, effective, and show promising transfer to the full BEIR and BRIGHT benchmarks as well as TREC DL 19 and 20. Our results suggest that evaluator-guided LLM program evolution is a practical path towards automatic discovery of novel ranking algorithms.

1 Introduction

Lexical retrieval algorithms such as BM25 [23] and query-likelihood (QL) [22] remain dominant first-stage rankers. Although learned sparse retrieval and dense retrieval methods have made significant advances [7, 9, 10], lexical methods retain key advantages in efficiency, interpretability, and strong out-of-domain robustness.

Over the decades, numerous variants have been proposed, including BM25T [8], BM25-adpt [18], BM25+ [19], BM25L [17], and alternative QL smoothing methods such as Dirichlet smoothing and Jelinek–Mercer [29]. Yet these improvements have largely relied on parameter tuning and human intuition about individual scoring components. This raises a natural question: can we automate the discovery of improved lexical retrieval algorithms? Recent developments in large language models (LLMs) for automated scientific discovery offer a promising path for such automation. Methods like FunSearch [24], followed by AlphaEvolve [21] represent candidate solutions as executable code and leverage LLMs to iteratively mutate and recombine programs, with performance-driven selection. A growing number of refinements and extensions have since been proposed within this paradigm [1, 11, 13–15]. These methods have produced strong results in mathematics and

*Work done while at Walmart Global Tech.
[2]Code: https://github.com/jmnian/RankEvolve.

AI Agents for Discovery in the Wild (AID-Wild), Workshop at ACM CAIS 2026.

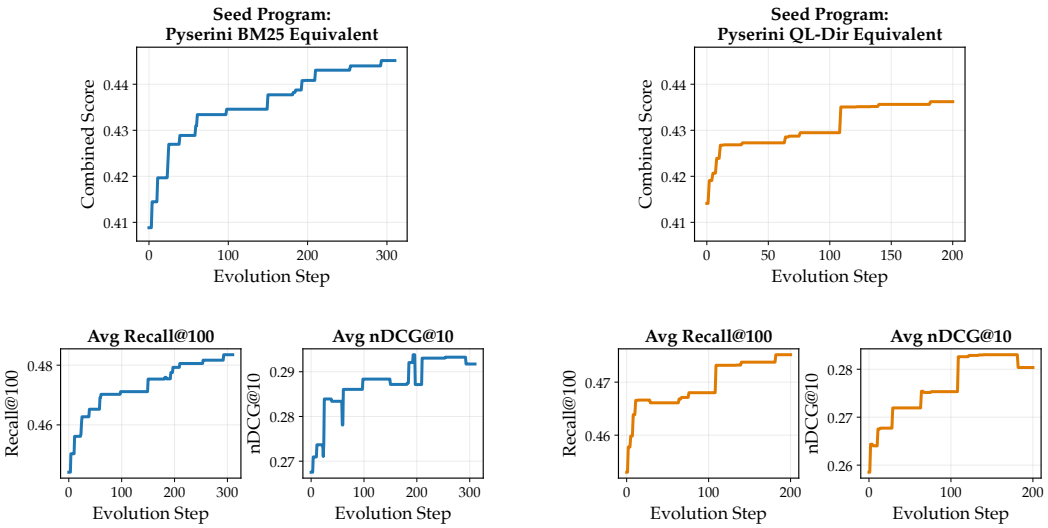

Figure 1: Evolution curves for two seed programs, with the optimization target decomposed into its components. The top row shows the best combined score over evolution steps for the BM25-equivalent and QL-Dirichlet-equivalent seeds. The bottom row shows the corresponding average Recall@100 and average nDCG@10 of the best-so-far programs. The combined score is defined as $0.8 \times$ Avg Recall@100 $+ 0.2 \times$ Avg nDCG@10, averaged across 12 IR datasets: BEIR (ArguAna, FiQA, NFCorpus, SciFact, SciDocs, TREC-COVID) and BRIGHT (Biology, Earth Science, Economics, Pony, StackOverflow, TheoremQA).

combinatorial optimization, but have not yet been applied to information retrieval. We introduce RankEvolve, a program evolution setup in which each candidate ranking algorithm is a self-contained Python program ($\sim$300 lines) scored by an evaluator across multiple retrieval datasets. The evaluator's per-dataset metrics, a single fitness score, sampled top-performing and inspirational candidates, and prior attempted changes together form the prompt for a coding LLM to propose the next mutation. Based on AlphaEvolve's description, the candidates are managed by a combination of MAP-Elites [20] and island-based evolutionary databases [26].

Starting from two seed programs, BM25 and query likelihood with Dirichlet smoothing, we evolve for several hundred steps. The resulting algorithms are novel and effective, introducing scoring mechanisms absent from either seed family. To assess generalization, we evaluate the evolved programs on held-out BEIR [28] and BRIGHT [25] subsets, as well as TREC DL 2019 [4], and DL 2020 [3], and find promising transfer beyond the datasets used during evolution. Our contributions are: (1) RankEvolve is the first attempt at evolving entire retrieval algorithms via LLM-guided program search; (2) we study the importance of seed program design, examining how structural freedom and abstraction affect the evolution outcome; and (3) we show that RankEvolve can discover algorithms with novel scoring features that transfer to unseen datasets, suggesting that evaluator-guided program evolution with LLMs is a promising way forward for automating IR research.

2 Related Work

The idea of automatically discovering ranking functions has been explored through genetic programming (GP). Fan et al. [6] proposed ARRANGER, a framework that uses GP to discover ranking functions from arithmetic operators ($+$, $\times$, $\log$) applied to IR features (tf, idf, dl). Cummins and O'Riordan [5] evolved local and global term-weighting schemes using a similar GP setup, producing functions that competed with BM25. A simpler alternative is grid search over existing hyperparameters [27], though this is limited to tuning a fixed function. RankEvolve differs in a fundamental way: classical GP evolves expression trees of arithmetic primitives by randomly swapping subtrees without understanding what the expression computes. RankEvolve uses an LLM as the mutation operator over a Python program, enabling reason-informed edits (e.g., recognizing that a signal for the document's coverage of the query is missing and introducing one) and a far richer search space. The

Table 1: Macro-averaged results on held-out BRIGHT (6), BEIR (8), and TREC DL 19/20 (2) datasets. Best per-group scores **bolded**. ⋆Evolved with RankEvolve. †Significant over the respective seed (per-query paired t-test, $p < 0.05$).

	BRIGHT		BEIR		TREC DL	
Method	nDCG@10	R@100	nDCG@10	R@100	nDCG@10	R@100
BM25 [23]	10.55	32.11	**48.16**	70.95	62.16	46.02
BM25+ [19]	9.72	31.54	45.71	69.93	61.07	46.69
BM25-adpt [18]	11.05	34.67	44.62	69.55	60.90	**48.23**
BM25⋆	**11.79**†	**37.51**†	47.90	**72.43**†	**64.57**†	47.10
QL-Dir [29]	9.52	32.48	44.15	68.72	57.03	46.69
QL-JM [29]	10.17	31.08	40.61	65.79	56.65	41.05
QL-Dir⋆	**11.42**†	**36.33**†	**46.46**†	**70.22**†	**60.68**†	**47.96**

learning-to-rank (LTR) paradigm [16] is also related, though the key distinction is that LTR combines existing features through learned weights, while RankEvolve proposes entirely new features and also evolves the relevance scoring function itself.

3 Method

We frame the discovery of improved retrieval algorithms as program synthesis via LLM-guided evolutionary search. RankEvolve iteratively mutates a seed program using an LLM and selects among variants based on retrieval performance. The overall pipeline consists of four components: (1) a seed program and a system prompt, which together define the search space; (2) a population database maintaining diversity via island-based evolution [26] and MAP-Elites [20]; (3) mutation guided by structured prompts; and (4) an evaluator that computes the fitness score over retrieval datasets.

3.1 Search Space

The seed program and system prompt jointly define the search space. The seed program defines the evolvable interface: code regions that the LLM is permitted to modify. To maximize evolutionary freedom, we decompose each ranking function into a small number of abstract components. For the BM25 seed, we define three evolvable components: query representation, document representation, and scoring function. The initial behavior reproduces classic BM25:

$$\text{BM25}(q,d) = \sum_{t \in q} \text{IDF}(t) \cdot \frac{\text{tf}(t,d) \cdot (k_1+1)}{\text{tf}(t,d) + k_1 \cdot \left(1 - b + b \cdot \frac{|d|}{\text{avgdl}}\right)} \tag{1}$$

$$\text{IDF}(t) = \log\left(\frac{N - \text{df}(t) + 0.5}{\text{df}(t) + 0.5}\right), \tag{2}$$

where $\text{tf}(t,d)$ is the term frequency of t in document d, $\text{df}(t)$ is the number of documents that contain t, $|d|$ is the document length, avgdl is the average document length, and $k_1 = 0.9$ and $b = 0.4$ following Pyserini [12] defaults. For the QL-Dir seed, we add a fourth component, the collection language model, to the evolvable interface. The initial program implements:

$$\text{QL-Dir}(q,d) = \sum_{t \in q} \log\left(\frac{\text{tf}(t,d) + \mu \cdot P(t \mid C)}{|d| + \mu}\right), \tag{3}$$

where $P(t \mid C) = \text{tf}(t,C)/|C|$ is the collection language model probability of term t, C is the corpus, and $\mu = 2000$ following Pyserini. The system prompt guides how to evolve the seed program. It details the design principles that encourage exploration of information-theoretic, probabilistic, and fundamentally novel ideas while discouraging ad-hoc constraints without justification. It also specifies the optimization objective, metrics, datasets, and evolvable versus restricted components. Together, these elements aim to maximize the LLM's freedom while ensuring every candidate remains a valid, executable retrieval system.

3.2 Population Management

We maintain a population of candidate programs using a combination of island-based evolution [26] and MAP-Elites [20]. For the island-based model, the population is partitioned into K independently evolving islands. Each island maintains its own MAP-Elites grid, including a record of its best programs. Programs inherit their parent's island, preserving lineage isolation. Within each island, MAP-Elites maps programs to a grid defined by two dimensions: complexity (code length) and diversity (edit distance from the population). Each dimension is divided into B bins, yielding $B \times B$ cells per island. A new program is accepted into a cell only if the cell is unoccupied or the new program has strictly higher optimization target score than the current occupant. This mechanism is crucial for avoiding local minima, because always picking the best-performing candidate for mutation would quickly stagnate the search. Every M iterations, the top γ fraction of programs from each island migrate to adjacent islands to encourage new variants. Programs that have migrated previously are excluded to prevent duplication.

3.3 Mutation Proposal

At each iteration a parent program is selected from the current island via one of three strategies, chosen at random: (1) exploration (probability p_e): uniform random sampling from the island; (2) exploitation (probability p_x): sampling from the MAP-Elite archive; or (3) weighted (probability $1 - p_e - p_x$): performance-proportional sampling from the island. Additionally, we include T best programs and S randomly sampled programs from the same island to provide the LLM with diverse reference points. Each program is paired with its detailed evaluation metrics described in Section 3.4. These, together with the system prompt, are presented to the optimizer LLM, which proposes a mutation in a `SEARCH/REPLACE` diff format.

3.4 Evaluator

The evaluator imports a candidate program, executes its full pipeline (tokenization, indexing, and retrieval) on a set of evaluation datasets, and returns per-dataset nDCG@10, Recall@100, and latency measurements, all of which are stored alongside the program in the population database. For population management, sampling priority, and as the optimization target, we use a single fitness score: $0.8 \times$ Avg Recall@100 $+ 0.2 \times$ Avg nDCG@10, where the averages are taken across datasets. The weighting reflects the first-stage retrieval setting. The primary objective is to maximize recall of relevant documents for a downstream reranker, while nDCG@10 serves as a secondary signal for ranking quality of the retrieved set.

4 Experiments

4.1 Setup

RankEvolve is implemented as an AlphaEvolve-style framework where candidates are managed using island-based evolution and a MAP-Elites archive. We use GPT-5.2 with medium reasoning effort via API, and set $K = 3$, $B = 12$, $M = 20$, $\gamma = 0.15$, $T = 4$, and $S = 4$ (Sections 3.2 and 3.3). RankEvolve ran for 300 steps from the BM25 seed and 200 steps from the QL-Dir seed, yielding the best-scoring programs at steps 293 and 182, respectively. Figure 1 shows the optimization trajectory over the course of evolution. A full 300-step BM25 run takes approximately 30 hours on a MacBook Pro, averaging 6 minutes per step; wall-clock time is dominated by evaluation across the 12 development datasets rather than LLM API calls.

We evaluate on three benchmark suites: 14 datasets from BEIR [28], all 12 subsets of BRIGHT [25], and TREC Deep Learning 2019 [4]/2020 [3], totaling 28 datasets. We exclude BioASQ, Signal-1M (RT), TREC-NEWS, and Robust04 as they are not publicly available on BEIR GitHub, and omit MSMARCO as it serves as the corpus for TREC DL. Of these 28 datasets, only 12 are used during evolution (from BEIR: ArguAna, FiQA, NFCorpus, SciFact, SciDocs, TREC-COVID; from BRIGHT: Biology, Earth Science, Economics, Pony, StackOverflow, TheoremQA). The remaining 16 are held out entirely and used solely to test generalization.

4.2 Results

Table 1 presents the macro-averaged results for each benchmark on the 16 held-out datasets. BM25* outperforms all BM25 baselines on BRIGHT and BEIR Recall@100 and achieves the highest nDCG@10 on TREC DL. QL-Dir* consistently outperforms both QL-Dir and QL-JM across all three benchmarks. Gains are statistically significant over the respective seeds on most evaluation groups, and extend to datasets not seen during evolution, indicating that RankEvolve discovers effective retrieval algorithms that generalize well rather than overfitting to the evaluator signal.

4.3 The Evolved BM25 Algorithm

After 293 evolution steps, the best evolved algorithm converges to a multi-channel, modulated scoring function that operates entirely over lexical features, yet has substantially departed from BM25 in structure. The top-level scoring function is:

$$\begin{aligned} S(q,d) = R(q^{\text{base}}, d^{\text{base}}) + w_{\text{pfx}} \cdot R(q^{\text{pfx}}, d^{\text{pfx}}) \\ + \; w_{\text{bi}} \cdot R(q^{\text{bi}}, d^{\text{bi}}) + w_{\text{mic}} \cdot G(q) \cdot R(q^{\text{mic}}, d^{\text{mic}}), \end{aligned} \tag{4}$$

where R is a shared core scoring function applied across four parallel token spaces. Base tokenization uses the standard Lucene tokenizer. Prefix tokenization ($w_{\text{pfx}} = 0.1$) truncates each token to its first 5 characters, acting as a cheap stemming approximation. Bigram tokenization ($w_{\text{bi}} = 0.08$) concatenates consecutive token pairs. Micro tokenization ($w_{\text{mic}} = 0.12$) uses rolling character 3-grams for sub-word matching, gated by $G(q) = \sigma((\overline{\text{IDF}}(q) - 2.2)/1.0)$, a sigmoid of the query's mean IDF that activates sub-word matching only for rare or technical queries.

The shared scoring function R combines a base evidence term with a chain of bounded multipliers:

$$R(q,d) = \frac{\ln(1+E) \cdot B_{\text{cov}} \cdot B_{\text{spec}} \cdot B_{\text{coord}} \cdot B_{\text{anc}}}{B_{\text{len}}}, \tag{5}$$

$$E = \sum_{t \in M} w(t) \cdot \ln(1 + \text{tf}(t,d)), \tag{6}$$

where E accumulates weighted log-TF evidence over matched query terms M, and B_* are multipliers of the form $1 + \alpha \cdot (\cdot)$ with small α that modulate the score based on query-term coverage, topical specificity, term coordination, rare-term anchoring, and document length. Their individual effects are gentle, but their combined effect can be substantial. The outer $\ln(1+E)$ applies a second layer of saturation, making the scoring robust to outlier term frequencies through double log-compression. Full definitions are available in the accompanying code repository.

We highlight three aspects of the evolved design that are especially notable. First, the composite term weight:

$$w(t) = \text{qtf}(t,q)^{0.5} \cdot \text{IDF}(t) \cdot \left(\frac{\text{IDF}(t)}{\text{IDF}(t)+1}\right)^{0.6} \cdot \frac{\text{IDF}(t)}{\text{IDF}(t)+1.25} \tag{7}$$

multiplies three separate functions of IDF(t), which together suppress stopword-like terms (low IDF) while leaving rare terms nearly unaffected. RankEvolve has found a soft stopword filter without ever being told about stopwords. Second, the specificity multiplier B_{spec} uses point-wise mutual information between each query term and the candidate document to reward documents where matched terms appear with higher-than-expected frequency, an idea closely related to the collection language model in probabilistic retrieval. Third, the length dampener $B_{\text{len}} = 1 + 0.15 \cdot \ln(1 + (|d| + 1)/(\text{avgdl} + 1))$ replaces BM25's linear normalization with a gentler logarithmic form, aligning with findings from Lv and Zhai [19] that BM25 over-penalizes long documents.

RankEvolve arrived at all of these components through evolutionary search alone, without any explicit guidance toward them. The prefix channel approximates stemming, the multi-layered IDF weighting acts as a soft stopword filter, the PMI-based multiplier (B_{spec}) resembles a collection language model, and the logarithmic length normalization independently addresses a known BM25 weakness. To the best of our knowledge, no exact formulation in the existing literature matches the evolved scoring function, yet these well-studied concepts emerged naturally from a purely metric-driven search process. These concepts and ideas are almost certainly present in the LLM's training data. However, the fact that they emerged from a purely metric-driven search, without any explicit guidance, suggests they may reflect fundamental properties of the lexical retrieval problem rather than incidental design choices.

Table 2: Ablation on seed-program structure (BM25 family). Optimization target defined in Section 3.4, evaluated on the 12 development (seen) and 16 held-out (unseen) datasets.

	Optimization Target	
Code Structure	Seen (12)	Unseen (16)
Original (step 0)	40.30	49.77
Constrained (step 115/200)	42.87	50.47
Composable (step 185/200)	43.31	51.33
Freeform (step 177/200)	44.35	51.20
Freeform (step 293/300)	**44.58**	**52.22**

4.4 The Evolved Query Likelihood Algorithm

After 182 evolution steps starting from classical query likelihood with Dirichlet smoothing, the best evolved algorithm retains the probabilistic language-modeling foundation but departs substantially from the standard formulation. The top-level scoring function is:

$$S(q,d) = \sum_{t \in q_u} \omega(t)\, \tilde{s}(t,d) \;+\; \sum_{t \in q_u} m(t,d) \;+\; \text{AND}(q,d) \;+\; \text{LP}(d), \tag{8}$$

where q_u is the set of unique query terms. The score decomposes into a weighted sum of per-term relevance scores $\tilde{s}$, a missing-term penalty m, a soft-AND coverage bonus, and a log-normal document length prior. Unlike the evolved BM25, which restructures scoring into a product of multipliers (Equation 5), the evolved QL retains an additive structure. We highlight four key departures from the seed. First, the standard collection language model $P(t \mid C)$ is replaced by a three-stage enriched estimate: raw probabilities are raised to a power $\tau = 0.85$ and renormalized to flatten the distribution toward rare terms, then interpolated with a document frequency model $P_{\text{df}}(t) = \text{df}(t)/N$, and finally mixed with a uniform floor. RankEvolve has independently discovered that flattening the collection language model improves retrieval, an idea related to information-based retrieval models [2]. Second, raw TF is replaced by $\text{tf}(t,d)^{\beta(t)}$ where $\beta(t) \in [0.70, 1.0]$ is a function of normalized IDF, providing per-term adaptive saturation that is more expressive than BM25 and the evolved BM25's uniform saturation. Third, a leaky rectifier keeps negative per-term scores at 12% strength, paired with a 7% missing term penalty for absent query terms, creating a layered penalty structure that standard QL lacks. Fourth, the length prior $\text{LP}(d) = -0.06 \cdot (\log(|d|) - \log(\text{avgdl}))^2$ penalizes deviation from average length in both directions, unlike most length penalties that only penalize long documents. Overall, the evolved QL program remains recognizably probabilistic, but replaces the seed's single smoothing mechanism with several interacting forms of adaptive smoothing, missing-term control, and document-length priors.

4.5 Ablation Study

We study how the structural freedom of the seed program shapes evolution. We design three functionally identical BM25 seeds with increasing degrees of freedom, and evolve each using the procedure from Section 3. The "constrained" seed fixes the BM25 formula and permits only hyperparameter tuning and selection among predefined component variants (e.g., Lucene vs. Robertson IDF). This setting is analogous to automated grid search. The "composable" seed decomposes retrieval into modular primitives whose formulas may be rewritten or extended, but keeps the overall pipeline structure fixed. The "freeform" seed, used in our main experiments, defines only query representation, document representation, and scoring function. It fixes the evaluator interface and leaves everything else evolvable.

Table 2 shows a clear trend: greater structural freedom yields monotonically higher optimization target scores on both the 12 datasets used during evolution and the 16 held-out datasets. Constrained evolution converges earliest but produces the smallest gains, confirming that parameter tuning alone has limited potential. The composable variant improves further by introducing novel scoring primitives, yet its fixed pipeline prevents deeper architectural changes. The freeform variant converges last but achieves the best scores on both seen and unseen datasets, demonstrating again that the improvements generalize rather than overfit to the development suite. These results suggest that seed program design meaningfully influences the effectiveness of the final solution discovered by

Table 3: Average per-document indexing latency and per-query retrieval latency across all 28 datasets. Lowest values are **bolded**, second lowest are underlined. ⋆Evolved with RankEvolve.

Method	Indexing (ms/doc)	Query (ms/query)
BM25	1.79	**56.72**
BM25⋆ Constrained	1.85	58.50
BM25⋆ Composable	**1.77**	111.49
BM25⋆ Freeform (step 177)	2.37	171.52
BM25⋆ Freeform (step 293)	2.81	648.89
QL-Dir	2.02	**178.26**
QL-JM	**1.95**	212.24
QL-Dir⋆ Freeform (step 182)	2.02	325.41

RankEvolve. Restrictive seeds bias the search towards local optima near the original formulation, whereas seeds with more structural freedom expand the search space and allow evolution to identify non-obvious improvements.

4.6 Latency Analysis

Table 3 reports per-document indexing and per-query retrieval latency averaged across all 28 datasets. Indexing overhead is negligible across all variants. Query latency increases with program complexity: the best performing BM25⋆ (step 293) is roughly $11\times$ slower than the seed BM25. This reflects the fact that no efficiency pressure was present during evolution. While latency statistics are visible to the LLM during evolution, they are never part of the optimization target nor explicitly mentioned to be mindful of in the system prompt, so the search process has no pressure to favor efficient solutions. This motivates the exploratory latency-aware experiment in Section 5, where we test whether query latency can be incorporated into the optimization target alongside retrieval effectiveness.

Despite this, the latency profile offers a useful lens into how RankEvolve discovers improvements at different levels of structural freedom. The constrained variant adds virtually no overhead (58.50 vs. 56.72 ms/query), confirming that its gains stem almost entirely from parameter tuning rather than algorithmic breakthrough. Yet even this minimal-complexity evolution meaningfully improves the optimization target (Table 2), showing that the simplest form of RankEvolve, which is effectively an automated parameter search, already provides value. Among the freeform variants, the trajectory from step 177 to step 293 is particularly informative. By step 177, the program had already achieved strong effectiveness (Table 2), while keeping query latency within a modest $3\times$ of the baseline. The following 116 steps of evolution continued to improve Recall and nDCG, but at the cost of a further $3.8\times$ increase in query latency. This suggests that later evolutionary steps increasingly exploit more complex scoring mechanisms whose marginal effectiveness gains come with disproportionate computational cost. In other words, the evolution exhibits a diminishing-returns pattern where earlier mutations capture high-impact structural improvements, and later mutations add refinements that are marginally effective but quite expensive. For practical deployment, the step 177 variant may offer a stronger effectiveness-efficiency tradeoff, achieving most of the effectiveness gains at roughly $3\times$ the baseline latency. More broadly, this observation suggests that RankEvolve is useful not only for identifying a single best-performing program, but also for exploring a family of retrieval algorithms that span different tradeoffs between effectiveness and efficiency.

5 Exploratory Study: Latency-Aware Evolution

We also conduct a small exploratory study to test whether RankEvolve can optimize retrieval effectiveness and query latency jointly. We run this study from the BM25 seed for 50 evolution steps. All other settings remain the same as in the main experiment, except we allocate part of the optimization target to query latency and we use Recall@1000 instead of Recall@100. Let $\mathcal{D}$ denote the set of 12 optimization datasets, p denote a candidate program, and p_0 denote the seed program. For each dataset $d \in \mathcal{D}$, let $t_d(p)$ be the median query latency of program p. We measure the latency

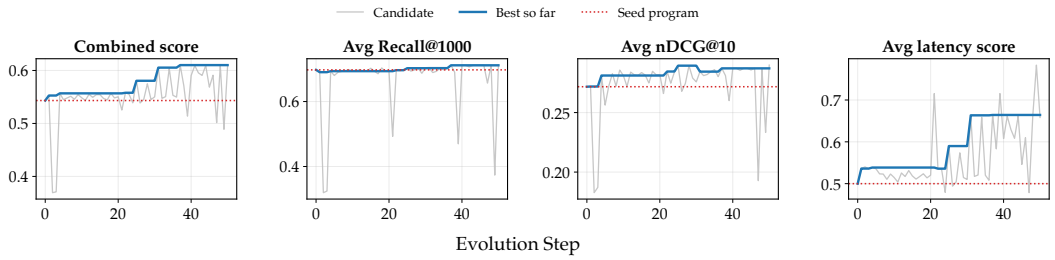

Figure 2: Latency-aware evolution across optimization steps. RankEvolve optimizes a weighted objective combining Recall@1000, nDCG@10, and a latency score measured relative to the seed program. The best-so-far candidate improves the combined score while increasing the latency-score component, indicating better query efficiency under the relative latency objective.

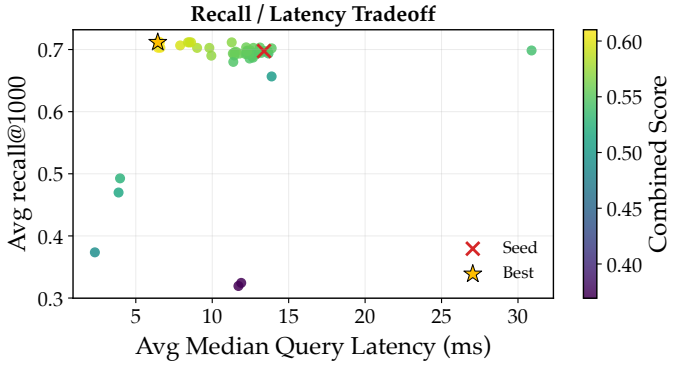

Figure 3: Recall-latency tradeoff for latency-aware evolution. Each candidate is plotted by average median query latency and average Recall@1000, with color indicating the combined score. The evolved candidates move toward better recall-latency tradeoffs compared with the seed.

of p relative to the seed by

$$\rho_d(p) = \frac{t_d(p)}{t_d(p_0)}.$$

Thus, $\rho_d(p) < 1$ means that p is faster than the seed on dataset d, while $\rho_d(p) > 1$ means that it is slower. We convert this relative latency into a bounded latency score:

$$\ell_d(p) = \frac{1}{1 + \rho_d(p)}.$$

This score is 0.5 when p matches the seed latency, greater than 0.5 when it is faster, and less than 0.5 when it is slower. We average this score across datasets:

$$L(p) = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \ell_d(p).$$

The final latency-aware objective is

$$J(p) = 0.45 \cdot R(p) + 0.20 \cdot N(p) + 0.35 \cdot L(p),$$

where $R(p)$ is average Recall@1000, $N(p)$ is average nDCG@10, and $L(p)$ is the average latency score across the 12 optimization datasets. Since this is an exploratory study, we choose the weights heuristically rather than tuning them. Candidates with $\max_{d \in \mathcal{D}} \rho_d(p) > 5$ receive a latency penalty, which discourages programs that improve retrieval quality only by becoming much slower.

Figure 2 and Figure 3 show that RankEvolve improves the multi-objective score while reducing average median query latency. This suggests that the evaluator can steer evolution toward more efficient retrieval programs. However, this study is only run on the optimization datasets and is not evaluated on held-out datasets, so we treat it as preliminary evidence rather than a generalization result.

6 Discussion and Limitations

During evolution, the evaluator returns both indexing and per-query latency, and these are surfaced to the optimizer LLM in the prompt. However, latency is deliberately left out of the optimization target score so that the program search is pushed purely toward retrieval effectiveness. We want to test whether RankEvolve can discover retrieval algorithms that genuinely improve held-out effectiveness, before asking it to manage effectiveness-efficiency tradeoffs. Therefore, the evolved programs should not be interpreted as efficiency-aware solutions. Additionally, we only studied a single evolutionary framework (AlphaEvolve), a single optimizer LLM (GPT-5.2), and a fixed evaluation set. Each of these choices may shape the search trajectory and the kinds of solutions that emerge.

GPT-5.2 was almost certainly trained on prior IR literature and perhaps also the broader benchmark ecosystem used here, so we do not claim RankEvolve achieves fully independent discovery. However, human research also builds on prior work, memory, analogy, and recombination, so it is not obvious that LLM-based discovery should be dismissed simply because the optimizer has seen related literature. Our claim is narrower: the evaluator verifies that the evolved programs improve retrieval effectiveness and generalize to held-out datasets. Several mechanisms in the evolved programs resemble ideas from prior work, but their exact formulations do not match any published algorithm we are aware of. In that sense, RankEvolve does not establish pure discovery, but it does provide evidence that LLM-guided evolutionary search can synthesize nontrivial lexical retrieval features.

7 Conclusion and Future Work

We introduce RankEvolve, a framework that applies LLM-guided program evolution to discover lexical retrieval algorithms. Evolved from BM25 and QL-Dir seeds, the resulting functions consistently outperform their seeds and established variants on held-out benchmark datasets. The evolved programs independently rediscover and reformulate well-studied IR concepts. Our ablation shows that the structural freedom of the seed program meaningfully affects the effectiveness of the final converged solution. Our exploratory latency-aware study suggests that efficiency can be incorporated into the evaluator, but future work should test whether such efficiency-aware programs generalize to held-out datasets under the same multi-objective setting. The evolved algorithms, while effective, are much more complex than their seeds. Defining and encouraging elegance is another promising direction, and may in turn elicit simpler and more fundamental solutions. More broadly, RankEvolve optimizes the objective specified by the evaluator, suggesting that the framework could extend beyond lexical retrieval to dense retrieval, learned sparse retrieval, late interaction methods, and LLM-based reranking. We hope RankEvolve motivates further exploration of LLM-guided program evolution as a tool for automated IR research.

References

[1] Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, Alex Dimakis, and Ion Stoica. Adaevolve: Adaptive LLM driven zeroth-order optimization. *CoRR*, abs/2602.20133, 2026. doi: 10.48550/ARXIV.2602.20133. URL https://doi.org/10.48550/arXiv.2602.20133.

[2] Stéphane Clinchant and Eric Gaussier. Information-based models for ad hoc ir. In *Proceedings of the 33rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '10, page 234–241, New York, NY, USA, 2010. Association for Computing Machinery. ISBN 9781450301534. doi: 10.1145/1835449.1835490. URL https://doi.org/10.1145/1835449.1835490.

[3] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, and Daniel Campos. Overview of the TREC 2020 deep learning track. In Ellen M. Voorhees and Angela Ellis, editors, *Proceedings of the Twenty-Ninth Text REtrieval Conference, TREC 2020, Virtual Event [Gaithersburg, Maryland, USA], November 16-20, 2020*, volume 1266 of *NIST Special Publication*. National Institute of Standards and Technology (NIST), 2020. URL https://trec.nist.gov/pubs/trec29/papers/OVERVIEW.DL.pdf.

[4] Nick Craswell, Bhaskar Mitra, Emine Yilmaz, Daniel Campos, and Ellen M. Voorhees. Overview of the trec 2019 deep learning track. In *Proceedings of the Twenty-Eighth Text REtrieval Conference (TREC 2019)*. National Institute of Standards and Technology (NIST), 2020. URL https://arxiv.org/abs/2003.07820.

[5] Ronan Cummins and Colm O'Riordan. Evolving local and global weighting schemes in information retrieval. *Inf. Retr.*, 9(3):311–330, 2006. doi: 10.1007/S10791-006-1682-6. URL https://doi.org/10.1007/s10791-006-1682-6.

[6] Weiguo Fan, Michael D. Gordon, and Praveen Pathak. A generic ranking function discovery framework by genetic programming for information retrieval. *Inf. Process. Manag.*, 40(4):587–602, 2004. doi: 10.1016/J.IPM.2003.08.001. URL https://doi.org/10.1016/j.ipm.2003.08.001.

[7] Thibault Formal, Benjamin Piwowarski, and Stéphane Clinchant. *SPLADE: Sparse Lexical and Expansion Model for First Stage Ranking*, page 2288–2292. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450380379. URL https://doi.org/10.1145/3404835.3463098.

[8] Mathias Géry and Christine Largeron. Bm25t: a BM25 extension for focused information retrieval. *Knowl. Inf. Syst.*, 32(1):217–241, 2012. doi: 10.1007/S10115-011-0426-0. URL https://doi.org/10.1007/s10115-011-0426-0.

[9] 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 Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu, editors, *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing, EMNLP 2020, Online, November 16-20, 2020*, pages 6769–6781. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.EMNLP-MAIN.550. URL https://doi.org/10.18653/v1/2020.emnlp-main.550.

[10] Omar Khattab and Matei Zaharia. Colbert: Efficient and effective passage search via contextualized late interaction over BERT. In Jimmy X. Huang, Yi Chang, Xueqi Cheng, Jaap Kamps, Vanessa Murdock, Ji-Rong Wen, and Yiqun Liu, editors, *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, SIGIR 2020, Virtual Event, China, July 25-30, 2020*, pages 39–48. ACM, 2020. doi: 10.1145/3397271.3401075. URL https://doi.org/10.1145/3397271.3401075.

[11] Robert Tjarko Lange, Yuki Imajuku, and Edoardo Cetin. Shinkaevolve: Towards open-ended and sample-efficient program evolution, 2025. URL https://arxiv.org/abs/2509.19349.

[12] Jimmy Lin, Xueguang Ma, Sheng-Chieh Lin, Jheng-Hong Yang, Ronak Pradeep, and Rodrigo Nogueira. Pyserini: A python toolkit for reproducible information retrieval research with sparse and dense representations. In Fernando Diaz, Chirag Shah, Torsten Suel, Pablo Castells, Rosie Jones, and Tetsuya Sakai, editors, *SIGIR '21: The 44th International ACM SIGIR Conference on Research and Development in Information Retrieval, Virtual Event, Canada, July 11-15, 2021*, pages 2356–2362. ACM, 2021. doi: 10.1145/3404835.3463238. URL https://doi.org/10.1145/3404835.3463238.

[13] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language model. In Ruslan Salakhutdinov, Zico Kolter, Katherine A. Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp, editors, *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, volume 235 of *Proceedings of Machine Learning Research*, pages 32201–32223. PMLR / OpenReview.net, 2024. URL https://proceedings.mlr.press/v235/liu24bs.html.

[14] Fei Liu, Yiming Yao, Ping Guo, Zhiyuan Yang, Xi Lin, Zhe Zhao, Xialiang Tong, Kun Mao, Zhichao Lu, Zhenkun Wang, Mingxuan Yuan, and Qingfu Zhang. A systematic survey on large language models for algorithm design. *ACM Comput. Surv.*, 58(8), February 2026. ISSN 0360-0300. doi: 10.1145/3787585. URL https://doi.org/10.1145/3787585.

[15] Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z. Pan, Alexander Du, Kurt Keutzer, Alexandros G. Dimakis, Koushik Sen, Matei Zaharia, and Ion Stoica. Evox: Meta-evolution for automated discovery. *CoRR*, abs/2602.23413, 2026. doi: 10.48550/ARXIV.2602.23413. URL https://doi.org/10.48550/arXiv.2602.23413.

[16] Tie-Yan Liu. *Learning to Rank for Information Retrieval*. Springer, 2011. ISBN 978-3-642-14266-6. doi: 10.1007/978-3-642-14267-3. URL https://doi.org/10.1007/978-3-642-14267-3.

[17] Yuanhua Lv and ChengXiang Zhai. When documents are very long, bm25 fails! In *Proceedings of the 34th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '11, page 1103–1104, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307574. doi: 10.1145/2009916.2010070. URL https://doi.org/10.1145/2009916.2010070.

[18] Yuanhua Lv and ChengXiang Zhai. Adaptive term frequency normalization for bm25. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 1985–1988, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307178. doi: 10.1145/2063576.2063871. URL https://doi.org/10.1145/2063576.2063871.

[19] Yuanhua Lv and ChengXiang Zhai. Lower-bounding term frequency normalization. In *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, CIKM '11, page 7–16, New York, NY, USA, 2011. Association for Computing Machinery. ISBN 9781450307178. doi: 10.1145/2063576.2063584. URL https://doi.org/10.1145/2063576.2063584.

[20] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *CoRR*, abs/1504.04909, 2015. URL http://arxiv.org/abs/1504.04909.

[21] Alexander Novikov, Ngân Vũ, Marvin Eisenberger, Emilien Dupont, Po-Sen Huang, Adam Zsolt Wagner, Sergey Shirobokov, Borislav Kozlovskii, Francisco J. R. Ruiz, Abbas Mehrabian, M. Pawan Kumar, Abigail See, Swarat Chaudhuri, George Holland, Alex Davies, Sebastian Nowozin, Pushmeet Kohli, and Matej Balog. Alphaevolve: A coding agent for scientific and algorithmic discovery, 2025. URL https://arxiv.org/abs/2506.13131.

[22] Jay M. Ponte and W. Bruce Croft. A language modeling approach to information retrieval. *SIGIR Forum*, 51(2):202–208, 2017. doi: 10.1145/3130348.3130368. URL https://doi.org/10.1145/3130348.3130368.

[23] Stephen E. Robertson and Hugo Zaragoza. The probabilistic relevance framework: BM25 and beyond. *Found. Trends Inf. Retr.*, 3(4):333–389, 2009. doi: 10.1561/1500000019. URL https://doi.org/10.1561/1500000019.

[24] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nat.*, 625(7995):468–475, 2024. doi: 10.1038/S41586-023-06924-6. URL https://doi.org/10.1038/s41586-023-06924-6.

[25] Hongjin Su, Howard Yen, Mengzhou Xia, Weijia Shi, Niklas Muennighoff, Han-yu Wang, Haisu Liu, Quan Shi, Zachary S. Siegel, Michael Tang, Ruoxi Sun, Jinsung Yoon, Sercan Ö. Arik, Danqi Chen, and Tao Yu. BRIGHT: A realistic and challenging benchmark for reasoning-intensive retrieval. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025. URL https://openreview.net/forum?id=ykuc5q381b.

[26] Reiko Tanese, John H. Holland, and Quentin F. Stout. *Distributed genetic algorithms for function optimization*. PhD thesis, USA, 1989. AAI9001722.

[27] Michael Taylor, Hugo Zaragoza, Nick Craswell, Stephen Robertson, and Chris Burges. Optimisation methods for ranking functions with multiple parameters. In *Proceedings of the 15th ACM international conference on Information and knowledge management*, pages 585–593, 2006.

[28] Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. BEIR: A heterogeneous benchmark for zero-shot evaluation of information retrieval models. In Joaquin Vanschoren and Sai-Kit Yeung, editors, *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*, 2021. URL https://datasets-benchmarks-proceedings.neurips.cc/paper/2021/hash/65b9eea6e1cc6bb9f0cd2a47751a186f-Abstract-round2.html.

[29] Chengxiang Zhai and John Lafferty. A study of smoothing methods for language models applied to information retrieval. *ACM Transactions on Information Systems (TOIS)*, 22(2):179–214, 2004.