The Case For Language Model Approximated LIKE Predicate

YINGZE LI, Harbin Institute of Technology, China
DONG WANG, Harbin Institute of Technology, China
ZIXUAN WANG, Harbin Institute of Technology, China
YINGLI ZHOU, The Chinese University of Hong Kong, Shenzhen, China
YU YAN, Harbin Institute of Technology, China
JIAN GENG, Harbin Institute of Technology, China
XINYUE WANG, Harbin Institute of Technology, China
ZIQING ZENG, Harbin Institute of Technology, China
HONGZHI WANG*, Harbin Institute of Technology, China

Modern databases often use the LIKE predicate to search text data. However, when the search condition is interrupted by wildcards, the existing search structure can degrade to a worst-case complexity linear to full table scale, resulting in poor performance. Traditional methods, such as B+-trees, fail to handle wildcards at both ends efficiently. Recent advances in language models offer a promising solution. These models can decode complex LIKE patterns into a small set of candidate values, which are then verified in dataset-size-invariant time via hash table lookups, greatly improving efficiency. However, integrating LLMs into databases faces challenges such as high latency, large storage requirements, and sensitivity to data distribution drifts.

To address these issues, we propose SMILE, a Small language Model Integrated LIKE Engine that learns column-local character distributions through small but exquisite parameters. Our SMILE acts as a neural translator that converts complex LIKE patterns into their corresponding result sets. Our approach achieves asymptotic complexity improvements while preserving SQL LIKE logic. We conduct comprehensive evaluation across diverse datasets to validate the efficacy of our approach. Our compact SMILE, with a parameter size 5 orders of magnitude smaller than large language models, achieves strong LIKE decoding efficiency and quality. Specifically, SMILE obtains high recall ability while accelerating LIKE by 3 orders of magnitude compared to large language models and sequential scans, 1.8-41.6 times faster than trigram indexes, and 2 orders of magnitude faster than B+-trees. Moreover, our model demonstrates robustness against potential data and query distribution drifts.

CCS Concepts: • **Information systems** → **Query optimization**.

Additional Key Words and Phrases: Query Optimization, LIKE Predicate, Language Models

ACM Reference Format:
Yingze Li, Dong Wang, Zixuan Wang, Yingli Zhou, Yu Yan, Jian Geng, Xinyue Wang, Ziqing Zeng, and Hongzhi Wang. 2026. The Case For Language Model Approximated LIKE Predicate. *Proc. ACM Manag. Data* 4, 1, Article 89 (February 2026), 30 pages. https://doi.org/10.1145/3786703

*Corresponding author.

Authors' Contact Information: Yingze Li, Harbin Institute of Technology, China, 23b903046@stu.hit.edu.cn; Dong Wang, Harbin Institute of Technology, China, 2021111049@stu.hit.edu.cn; Zixuan Wang, Harbin Institute of Technology, China, 2023113027@stu.hit.edu.cn; Yingli Zhou, The Chinese University of Hong Kong, Shenzhen, China, yinglizhou@link.cuhk.edu.cn; Yu Yan, Harbin Institute of Technology, China, yuyan@hit.edu.cn; Jian Geng, Harbin Institute of Technology, China, gengj@stu.hit.edu.cn; Xinyue Wang, Harbin Institute of Technology, China, 2022113365@stu.hit.edu.cn; Ziqing Zeng, Harbin Institute of Technology, China, 2024111701@stu.hit.edu.cn; Hongzhi Wang, Harbin Institute of Technology, China, wangzh@hit.edu.cn.

ACM 2836-6573/2026/2-ART89
https://doi.org/10.1145/3786703

1 INTRODUCTION

Modern database systems heavily rely on the LIKE predicate to efficiently retrieve and compute relevant textual information in relational tuples [7, 50, 59, 60]. This operator is crucial for various applications, including substring matching [53], data cleaning [36, 68, 105], and entity recognition [26].

Despite the widespread applications of the LIKE predicate, optimizing it remains a formidable challenge. Traditional B+-tree indexes work well for prefix/suffix patterns [23, 38] but struggle with wildcard-surrounded queries such as "%keyword%" [29, 96]. While trigram indexes support substring searches, they have high storage costs and limited wildcard flexibility [29]. When the queried keyword is interrupted by wildcards, or when the retrieving keyword is a "stop-word" present in all tuples [31], retrieval efficiency drops to $O(N \times \ell)$, where N represents the filtered table size, ℓ is the average string length. Furthermore, trigram indexes are not universally supported across databases [33, 75]. The fundamental limitation of existing approaches lies in their reliance on partial unmasked segments of the LIKE pattern for filtering and search operations, neglecting the holistic context of the query. This constraint significantly undermines their efficacy, particularly when dealing with complex wildcard patterns.

While exact evaluation for arbitrary LIKE patterns inherently requires $O(N\times\ell)$ complexity due to unavoidable full scans for high-cardinality cases (e.g., pure "%"), a critical class of interactive applications requires only a small set of guaranteed matches with sub-second latency. For instance, in data cleaning, in validating Pattern Functional Dependencies (PFDs) [78] such as [Name LIKE "%John%"] $\rightarrow$ [gender = "M"], within a human-in-the-loop workflow [14, 35, 63]. In this setting, prohibitive scan latencies disrupt the interactive loop, while IR-style approximations are unsuitable as they return logically incorrect matches (e.g., "Joan"). This dilemma motivates an approach rooted in the principles of Approximate Query Processing [15, 76], where trading exhaustive completeness for interactive speed is a well-accepted compromise. We therefore propose a generate-and-validate strategy. If we have a near-oracle predictor to generate a constant-sized candidate set, we can then verify them in $O(\ell)$ time. This eliminates the dependency on dataset size N, delivering the performance expected in interactive systems that present constant-sized result sets [9, 17, 18, 49, 61]. We therefore investigate the feasibility of constructing such an oracle for efficient LIKE query processing.

Fortuitously, Large Language Models (LLMs) [13, 79, 80] approximately satisfy this oracle requirement. Thanks to their pre-trained masked prediction capabilities [27, 69, 89], LLMs can infer missing tokens from incomplete wildcard patterns, making them perfectly suitable for decoding LIKE wildcards into candidates. Figure 1 illustrates the core idea of our approach. For a query such as `SELECT * FROM T WHERE Conference LIKE "%IGMO%"`, we may apply prompt engineering to guide the LLM in understanding that the Conference column contains academic venue names. LLMs can then infer that the pattern "%IGMO%" likely corresponds to the name SIGMOD. We can then verify the presence of SIGMOD via a fast $O(\ell)$ hash lookup. Building on this potential, we proceed to evaluate whether we can significantly reduce the worst-case $O(N \times \ell)$ runtime of such interactive LIKE queries by using language models as efficient autocomplete keyboards—transforming patterns into a fixed set of candidate strings which are verified in $O(\ell)$ time via hash-based lookups.

However, deploying LLMs to accelerate the LIKE predicate faces a huge obstacle. LLMs are too **large** to fit into conventional DBMS [34, 70]. It therefore brings three barriers: (1) **Latency gap between DB applications and LLMs:** The LLM's response generation incurs second-level delays per query, conflicting with database's demands. For most OLAP scenarios with medium-sized data, a full-table scan (1 second) may be completed before LLMs finish their first sentence (10 seconds). (2) **Storage overheads of LLMs:** The storage space occupied by LLMs is astonishing. For example, DeepSeek-V3, with 671 billion parameters, requires nearly 1TB of storage [93]. This

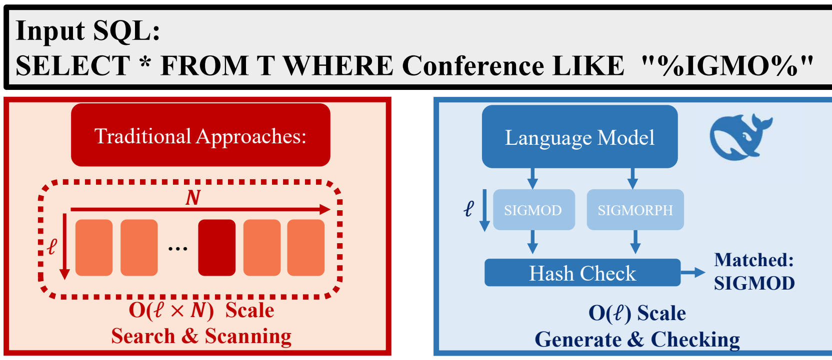


Fig. 1. Example for language model Approximated LIKE.

storage consumption can exceed the size of the entire database, significantly reducing the applicability of the method. (3) **Distribution drifts:** In database systems, data evolve continuously, necessitating frequent updates to model parameters [45, 66, 102]. However, fine-tuning LLMs is computationally expensive. To conclude, the "large" nature of existing LLMs presents challenges that hinder the direct deployment of translation LIKE predicates via LLMs. We have to explore whether it is possible to achieve similar LIKE translation functionality with a smaller model.

Fortunately, we observe that the distribution of characters within a single column can be regarded as a unique, special language with limited vocabulary combinations. This observation implies that translating LIKE templates into this column's sub-language can be learned via a small language model with limited parameters, obviating the need for resource-heavy LLMs. This insight motivates us to explore the use of lightweight Small Language Model(SLM) to learn the joint probability distribution of these characters. SLM, with smaller parameter scale, consumes significantly less memory and offers faster inference speed, making it well-suited for latency-sensitive database scenarios. Additionally, the lightweight nature of SLM makes it easier to manage through incremental finetuning in dynamic environments. Therefore, this observation inspires us to investigate whether we can **leverage small-scale language models to learn the language distribution of a single table** while retaining the functionality of LLMs for decoding LIKE predicate.

Based on the above discussions, we propose SMILE (Small language Model Integrated LIKE Engine), which leverages a lightweight character-level language model to translate LIKE queries into their result sets. This approach tackles the key challenges of LIKE predicate optimization. SMILE learns character distributions within specific column contexts, rather than across broad natural language domains. Coverage in this specific domain needs just a 16MB network, which is 5 orders of magnitude smaller than typical LLMs, yet it still translates LIKE patterns effectively. This small yet powerful model significantly boosts inference efficiency and reduces maintenance overhead. More importantly, SMILE transforms the inherently difficult LIKE matching problem into a more manageable string candidate verification task. During query optimization, SMILE generates candidate strings that match the LIKE pattern's logical constraints through autoregressive decoding. These candidates are then verified against the database using hash-based lookups. This hybrid approach converts the worst-case $O(N \times \ell)$ full table scans into $O(\ell)$ lookups for valid candidates, achieving substantial complexity improvements while preserving approximate SQL logic In summary, our work makes the following key contributions:

C1. A Novel Framework for LIKE Predicate Optimization: We introduce a new framework that recasts LIKE predicate optimization as a sequence translation task. This approach employs language models to directly generate candidate tuples from wildcard patterns, fundamentally shifting the execution paradigm from costly linear scans ($O(N \times \ell)$) to an efficient generate-and-validate process.

C2. Lightweight and Scalable SLM Architecture: To overcome the practical limitations of large language models (LLMs), we design and implement SMILE, a lightweight (16MB) and column-specific Small Language Model (SLM). SMILE acts as a specialized neural translator, efficiently learning data distributions to decode `LIKE` predicates into result sets. We further propose a hybrid strategy that enables our approach to scale to long text columns, which is a critical challenge for existing methods.
C3. Theoretical Framework with Probabilistic Guarantees: We establish a theoretical framework that provides probabilistic approximation guarantees for language-model-based `LIKE` decoding. By introducing the concept of pattern entropy to quantify query complexity, our framework formally bounds the sampling effort required to retrieve a target number of results with high confidence, establishing a principled link between model capacity, query structure, and performance.
C4. Evaluation Guided by Real-World Workload Analysis: To ensure a realistic evaluation, we first conducted a large-scale survey of 84,047 queries across 30 public databases to characterize real-world `LIKE` predicate usage. Guided by the identified patterns, we designed a representative benchmark on which SMILE achieves speedups of up to three orders of magnitude over sequential scans, and outperforms specialized trigram and B^+-tree indexes by 1.8-41.6× and two orders of magnitude, respectively, while using a model five orders of magnitude smaller than mainstream LLMs.

Roadmap: This paper is organized as follows. Section 2 formalizes the SQL `LIKE` predicate as a sequence-to-sequence task via neural machine translation. Section 3 evaluates state-of-the-art LLMs on `LIKE` pattern decoding, assessing accuracy, efficiency, and practical viability in database contexts. Section 4 introduces SMILE, a lightweight language model designed for efficient `LIKE` acceleration. In Section 5, we establish theoretical guarantees and extend the method's applicability. Section 6 presents experimental results comparing SMILE against baselines in accuracy, scalability, and robustness. Section 7 surveys related work, and Section 8 concludes with a summary, limitations, and future directions. The source code of SMILE and the Appendix are available at GitHub [4].

2 Preliminaries

In this section, we first provide an overview of the SQL `LIKE` predicate and language models in Section 2.1. Subsequently, we introduce a novel perspective on the optimization of the `LIKE` predicate from the viewpoint of neural language translation in Section 2.2. Section 2.3 explores practical interactive scenarios.

2.1 Basic Concepts

In this section, we introduce the SQL predicate `LIKE` and the definition of language models. Firstly, we delve into the definition of the `LIKE` predicate:

The `LIKE` predicate: The `LIKE` predicate facilitates the selection of tuples within a relational structure where attribute values match a specified pattern string. Formally, given a relation table T with attribute A, where the average string length of A is ℓ, and N tuples in total, the `LIKE` predicate $\mathcal{P}$ collects all tuples that satisfy $\mathcal{P}$ in result set $R_1 = \{t \in T \mid t[A] \text{ matches } \mathcal{P}\}$. As standardized in ISO/IEC 9075-2:2023 (Section 8.5, General Rule 3.b.ii) [1], the predicate recognizes two meta characters: (1) The percent symbol (%): Matches any substring of zero or more characters. (2) The underscore (_): Matches exactly one arbitrary character.

To illustrate the functionality of `LIKE` predicate, we provide the following examples. (E1): Pattern `LIKE "SIGMOD"` yields a condition where string matches `"SIGMOD"` string. (E2): Pattern `LIKE "SIG%"` matches all strings with the prefix `"SIG"` through left-anchored pattern matching. (E3): Pattern `LIKE "S_GMOD"` matches strings length of six with the first character being `"s"` and the last being `"GMOD"`.

Problem Definition: Given a `LIKE` predicate $\mathcal{P}$, we propose to optimize its approximate processing through dual objectives: minimizing query latency while maximizing recall of retrieved tuples. The recall metric is formally defined as: Recall $= \frac{|R_1 \cap R_2|}{|R_1|}$, where R_2 is the retrieved tuples, R_1 is $\mathcal{P}$'s ground truth result set.

Then, we proceed to provide the definition of language models.

Language model: Given vocabulary $V = \{x_1, x_2, \ldots, x_{|V|-2}, \langle SOS \rangle, \langle EOS \rangle\}$ and a sequence of preceding tokens $s_{<t} = (\langle SOS \rangle, x_{\pi 1}, x_{\pi 2}, \ldots, x_{\pi t-1})$, the language model $\mathcal{M}$ learns the conditional probability $\Pr(x_{\pi t} \mid x_{\pi 1}, x_{\pi 2}, \ldots, x_{\pi t-1})$ and samples the next token $x_{\pi t}$ based on this conditional probability distribution. That is:

$$x_{\pi t} \sim \Pr(x_{\pi t} \mid x_{\pi 1}, x_{\pi 2}, \ldots, x_{\pi t-1})$$

Given a fixed-length string input, the language model performs autoregressive decoding token-by-token iteratively. This iterative process begins with the start-of-sequence token $\langle SOS \rangle \in V$. In the t-th iteration, given the input $s_{<t} = (\langle SOS \rangle, x_{\pi 1}, x_{\pi 2}, \ldots, x_{\pi t-1})$, the language model generates the next token $x_{\pi t}$ based on $s_{<t}$. The generated token $x_{\pi t}$ is then appended to the historical input to form $s_{<t+1} = (s_{<t}, x_{\pi t})$. This updated string $s_{<t+1}$ serves as the input for the next iteration. The iterative process continues until $x_{\pi t}$ is the end-of-sequence token $\langle EOS \rangle \in V$.

2.2 LIKE Predicate Decoding: A Neuro-Linguistic Translation Perspective

We reconceptualize LIKE predicate evaluation as a specialized machine translation task. In this view, a LIKE pattern $\mathcal{P}$ is a source-language query that must be translated into its logical equivalents within the database's string domain (the target language). Formally, let A be the target column domain with N tuples. Conventional evaluation computes $\{s \in A \mid s \texttt{ matches } \mathcal{P}\}$ via deterministic approaches and faces a worst-case sequential scan for patterns like "%*keyword*%" in $O(N \times \ell)$ time. In contrast, our approach learns a translator $\mathcal{M} : \mathcal{P} \to \mathcal{A}" \subseteq A$, which translates the pattern into its results, where:

$$\mathcal{A}" = \arg\max_{s \in A} \Pr(s \mid \mathcal{P}) \quad \text{subject to} \quad s \texttt{ matches } \mathcal{P}$$

Here, $\Pr(s \mid \mathcal{P})$ represents the language model $\mathcal{M}$'s learned distribution. The verification condition $s \texttt{ matches } \mathcal{P}$ ensures the correctness of the result, while the generation process leverages $\mathcal{M}$'s logical understanding to bypass exhaustive comparisons. This dual-phase approach achieves sublinear complexity by restricting pattern matching to a probabilistically generated candidate set $|\mathcal{A}"| \ll |A|$.

This paradigm shift overcomes the fundamental limitations of traditional indexes through logic-aware holistic decoding. While trigram indexes fail to fetch wildcard-broken or "stop-word" patterns, and B+-trees stumble on non-prefix alignment, the translation language model exploits distributional semantics to directly infer lexically valid matches. Our experimental validation in Section 6 demonstrates that this approach maintains SQL-standard logic while reducing asymptotic complexity from $O(N \times \ell)$ to $O(\ell) + \mathcal{M}_{decode}$, where $\mathcal{M}_{decode}$ represents the decoding overhead of modern Transformer architectures.

2.3 The Case for Instantaneous, Approximate LIKE Applications

Instantaneous feedback on LIKE predicates presents a dilemma in modern data systems. Traditional query processing guarantees completeness via exhaustive scans but suffers from high latency. Conversely, information retrieval techniques—such as full-text indexing [65] or embedding based reranking [46, 58] do not preserve the logical correctness of SQL LIKE patterns. We argue for a third paradigm for interactive applications: one that trades completeness for speed while strictly preserving logical correctness. These applications need a small, guaranteed-correct subset in interactive time—a capability current systems lack.

Rule Validation in Interactive Data Cleaning: Interactive data cleaning [22, 68, 81] relies on rapid, human-in-the-loop validation [14, 35, 63] of rules like Pattern Functional Dependencies (PFDs) [78]. For instance, when validating a rule such as `[name LIKE "%John%"] -> [gender="M"]`, an analyst needs immediate examples to check the logic. A slow full scan disrupts this cognitive flow, while IR-style approximate matching returns logically irrelevant hits (e.g., Joan" or Johnny Appleseed") that obscure logical correctness. Here, the trade-off is clear: sacrificing completeness for the interactive speed needed to validate rules is a small price to pay. Our system delivers a small set of guaranteed-correct matches, preserving the tight human-in-the-loop process.

Fast Maintenance of Learned Cardinality Estimators: Learned cardinality estimators for `LIKE` predicates [7, 59, 88] must be continuously retrained on evolving data to mitigate "catastrophic forgetting" [39, 55]. A cornerstone of this adaptation is experience replay [82, 112], which necessitates frequent acquisition of ground-truth labels—i.e., actual query results—for diverse `LIKE` patterns from the current dataset. However, full-table scans to obtain these labels incur prohibitive latency, rendering online learning impractical. Our approach provides a remedy by generating a training-batch-sized, logically correct sample in sublinear time. For this task, completeness is unnecessary, but correctness is non-negotiable to avoid label noise, making the trade-off for speed ideal.

Pattern Induction in Information Extraction: Information Extraction (IE) relies on rapid "hypothesize-and-verify" cycles to refine patterns [12, 20, 110]. An engineer refining a pattern `LIKE "%John@%.edu%"` needs immediate feedback on its matches, but conventional query latency disrupts this iterative process. Our method provides millisecond-level, logically correct responses. Here, the goal is not completeness but to instantly receive a few logical witnesses to the pattern's behavior, allowing developers to validate and refine their logic without delay.

3 Large Language Models: Lessons Learned

In this section, we systematically evaluate the following hypothesis: *Since modern large language models(LLMs) have demonstrated remarkable zero-shot capabilities [30, 72, 103], could they serve as an "off-the-shelf" solution for decoding complex* `LIKE` *predicates, bypassing the need for specialized indexes or models?* In this section, we systematically investigate this possibility. We evaluate mainstream LLMs on both decoding quality and efficiency, treating them as a potential baseline. However, our findings reveal fundamental limitations that preclude their direct use: (1) substantial storage requirements, (2) prohibitive inference latency, and (3) a distributional mismatch leading to incomplete answers, which are unacceptable in a database context. These results thus serve as a crucial motivation, demonstrating that a specialized approach is not just an alternative, but a necessity.

Experimental Setup: We evaluate eight state-of-the-art open-source LLMs (ranging from 1.5B to 671B parameters) on `LIKE` predicate decoding. Our evaluation employs two real-world datasets: *TPC-H (lineitem)* [97], comprising 24M natural-language-like strings, and a recent snapshot of the *IMDB (name)* dataset [48], containing 15M real-world strings from the `primaryName` column.

To ensure practical relevance, our workload generation methodology is grounded in an analysis of 84,047 real-world `LIKE` patterns (see Appendix A). We focus on scenarios where conventional prefix/suffix indexes fail—specifically, patterns such as `LIKE "%Keyword%"` and `LIKE "%Key%Key%"`. We define two primary workloads:**W1 (Syllable-based):** Wildcards mask coherent phonetic syllables, simulating partial recall of word spellings where users reconstruct terms via pronunciation. **W2 (Morpheme-based):** Wildcards obscure morphemes—the smallest meaningful linguistic units—mimicking searches driven by semantic components. To enhance realism and complexity, we randomly replace 0 to 5 characters in the non-wildcard segments of each pattern with underscore ("_") wildcards. This process yields 2,000 test queries. Additional details are provided in Section 6.1.

Decoding Quality Analysis: We measure recall—the fraction of relevant strings correctly generated by each LLM for a given `LIKE` predicate. As illustrated in Figure 2, decoding accuracy

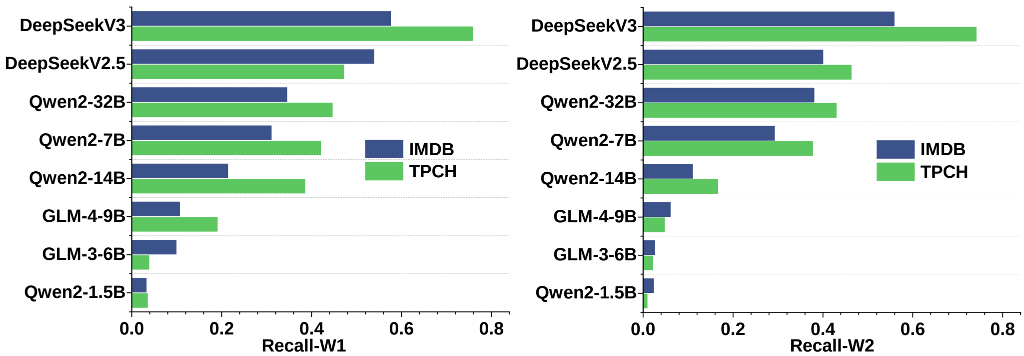


Fig. 2. LLM recall performance across datasets and workloads. Left: W1. Right: W2.

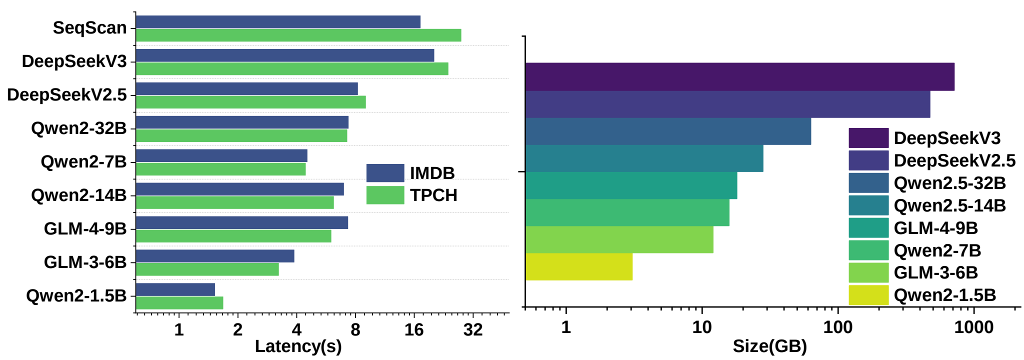


Fig. 3. Computational cost of LLMs. Left: Inference latency. Right: Model size.

correlates positively with model scale. However, a significant performance ceiling is evident, as no model approaches perfect recall. Even on the TPC-H dataset, where strings follow easier patterns, the state-of-the-art 671B-parameter model achieves a recall of only approximately 0.8. This persistent gap stems from a fundamental distributional mismatch, or more precisely, a bias mismatch. The open-world knowledge in an LLM's pre-training corpus instills an uncontrolled and undesirable bias towards general natural language. This generalist bias causes the LLM to generate numerous logically plausible strings that are valid in general language but do not exist within the target table's closed-world domain, thereby depressing recall. This issue is exacerbated on the IMDB dataset, whose structured strings deviate more sharply from natural language priors, leading to a more pronounced performance decline. Nevertheless, the core limitation persists even on linguistically coherent data, highlighting a systemic challenge in applying generalist models to specialized data retrieval tasks.

Computational Cost Analysis: Figure 3 quantifies the operational overhead of LLM-based decoding. High-performing models incur substantial latency; notably, DeepSeek-V3—the top performer in recall—exhibits inference times approaching those of a full sequential scan, rendering it unsuitable for latency-sensitive query processing. More critically, storage requirements pose an insurmountable barrier: state-of-the-art LLMs demand gigabytes to terabytes of parameter storage—often exceeding the size of the database itself. This inefficiency arises from the *generalist's burden*: LLMs are trained on diverse tasks (e.g., code generation [37], mathematical reasoning [111], dialogue [19]), resulting in a model whose parameters are overwhelmingly biased towards knowledge domains irrelevant to LIKE decoding. Although LLMs demonstrate zero-shot capability, their immense computational and storage demands are fundamentally incompatible with the efficiency constraints of database systems, where every CPU cycle and byte is carefully optimized.

Key Insights: *1. Foundational Decoding Capability:* LLMs can map wildcard patterns to logically plausible completions, achieving high recall (up to 80%) on linguistically coherent data by leveraging learned language distributions.

2. The Prohibitive Cost of Scale: High accuracy requires massive models, incurring severe latency and storage overheads—a "success tax" that renders direct deployment economically and operationally infeasible.

3. The Generalist's Bias: General-purpose knowledge introduces an uncontrolled and inefficient bias; most model capacity is consequently irrelevant to `LIKE` decoding, violating core database design principles of specialization and resource frugality.

These results offer a crucial insight. The initial enthusiasm for general-purpose LLMs is met by the harsh reality of the "generalist's burden"—their vast knowledge introduces uncontrolled bias and prohibitive overhead, making them impractical for production database use. This realization motivates a deliberate shift in design philosophy: we argue that a workload-aware, specialized solution is a more principled and efficient engineering choice. Consequently, we propose a lightweight small language model designed to overcome the exact limitations identified in this section.

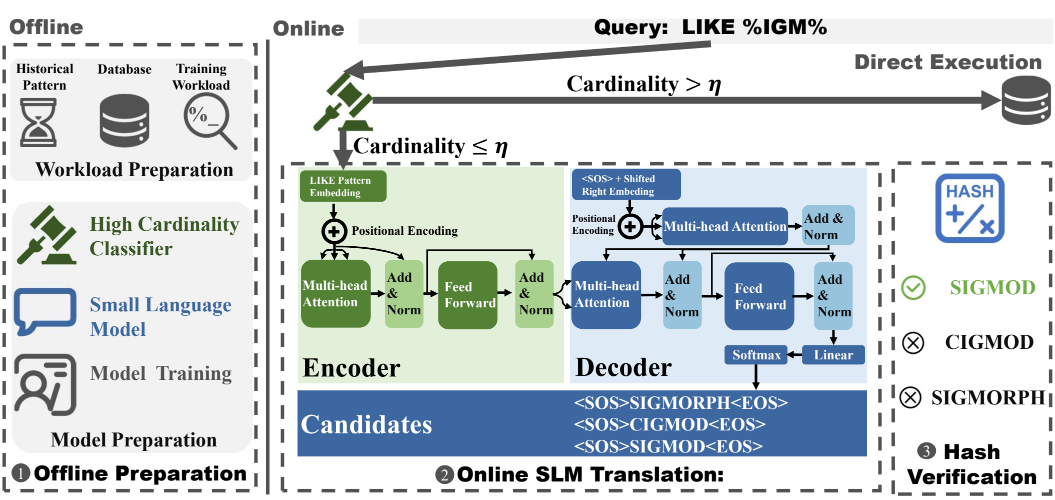


Fig. 4. Overview on SMILE.

4 Small Language Models: An Economically Viable Alternative

To overcome the limitations imposed by the LLMs' generalist bias, we propose SMILE (Small language Model Integrated LIKE Engine), a cost-effective solution that leverages SLMs to enhance the efficiency of `LIKE` predicate processing. SMILE is designed around a core principle: the intentional learning of a potent *inductive bias*. Instead of relying on open-world knowledge, SMILE learns the specific character-level distribution of a target column. This learned domain prior is the key to its efficiency and high recall, enabling it to act as a specialized neural translator that operates within the closed world of the database column.

4.1 SMILE Overview

As depicted in Figure 4, SMILE operates through three core stages: offline preparation of model and training workloads, online small language model driven translation of `LIKE` predicates, and hash-based validation of candidate strings. SMILE aims to optimize low-cardinality LIKE query patterns where dataset-size-independent decoding acceleration via SLM becomes feasible. This section provides a concise workflow overview. We formally introduce SMILE's SLM architecture and decoding strategies in Sections 4.2 and 4.3, respectively. Section 4.4 discusses decoding optimization techniques, while Section 4.5 details our query routing mechanism for identifying targetable low-cardinality queries.

1. Offline preparation: We initially prepare the training workloads offline. Subsequently, we train a classifier on the target column A and a character-level small language model using the prepared workloads. Inspired by prior works on learned database [86, 104], we first cluster historical queries into high-cardinality and low-cardinality types. We then train a binary classifier that classifies query patterns into these two categories. For low-cardinality patterns, we statistically model their wildcard distributions ("%" and "_" placements) to generate synthetic training data. We firstly sample N_s strings from A and obtain training sampling set $\mathcal{S} = \{s \mid s \in A\}, |\mathcal{S}| = N_s$. Then, we inject wildcards into the sampling set $\mathcal{S}$ to form supervised training information $\langle \mathcal{P}_i, s_i \rangle$ pairs where wildcard pattern $\mathcal{P}_i$ constitutes a source language and original tuples s_i form the target language. We employ teacher forcing with cross-entropy loss to train SMILE's SLM:

$$\mathcal{L} = -\sum_{\langle \mathcal{P}_i, s_i \rangle} \sum_{x_{ij} \in s_i} x_{ij} \log \mathcal{M}(\mathcal{P}_i)_j$$

Here, x_{ij} represents the true distribution of target string s_i's j-th token, and $\mathcal{M}(\mathcal{P}_i)_j$ represents the language model's output probability distribution for the j-th output token given wildcard input $\mathcal{P}_i$. The training approach mentioned above can be adapted for data updates. In scenarios where data undergoes periodic changes, we can similarly generate training samples and wildcard patterns from the new data distribution to fine-tune the model parameters. This training process enables SMILE to acquire its inductive bias. By learning exclusively from the target column's data, the model internalizes a domain-specific prior that constrains its predictive distribution to the statistical patterns of this closed world, effectively eliminating the out-of-domain generation that plagues generalist models.

2. Online SLM translation: For unknown queries, we first utilize a character-level language classifier to route testing query workloads. Subsequently, based on the classification outcomes, we meticulously select wildcard patterns with low-cardinality from the testing workloads. These low-cardinality wildcard patterns can be efficiently translated into candidate sets in $O(\ell)$ time. For these low-cardinality queries, we employ a neural character Transformer to translate the source `LIKE` patterns into target strings in the database. However, if the query has high-cardinality (e.g., pure "`LIKE %`"), even if language model can successfully decode such queries, the number of samples generated by the language model is proportional to the scale of the entire database, which restricts the space for the language-model-based optimization. Therefore, we directly deliver these query patterns for the database to execute.

3. Hash verification: After we obtained the constant scale candidate set via the generation of our SLM, we verified our candidate set using hash index, allowing results to be checked in $O(\ell)$ time.

4.2 Transformer: The Efficient LIKE Translator

We find the Transformer architecture [8, 51, 99], originally developed for machine translation, is exceptionally well-suited for our `LIKE` predicate decoding task. This suitability stems from two of its inherent properties:

(A1). Intrinsic Seq2Seq Architecture: The "%" wildcard implies that the length of a matching string is independent of the pattern's length. This input-output length mismatch renders fixed-size architectures, such as standard MLPs, unsuitable for learning this mapping [8, 92]. The Transformer's encoder-decoder structure naturally addresses this challenge. The encoder maps the variable-length input pattern to a latent representation $\mathbf{H} \in \mathbb{R}^{m \times d}$. The decoder then autoregressively generates the output sequence y_t conditioned on $\mathbf{H}$, terminating upon generating an $\langle EOS \rangle$

token. This mechanism effectively decouples the input and output lengths, making it ideal for the variable-length translation required by `LIKE` predicates.

(A2). Dynamic Attention Mechanism: When decoding a `LIKE` predicate, different segments of the input pattern have varying importance for generating the output string. The attention mechanism enables the model to dynamically focus on the most salient input tokens for each decoding step by adaptively weighting the input sequence. Formally, given an input sequence of d-dimensional embeddings $\mathbf{X} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_m)$, attention computes Query, Key, and Value matrices via learnable linear projections: $\mathbf{Q} = \mathbf{X}\mathbf{W}_Q, \quad \mathbf{K} = \mathbf{X}\mathbf{W}_K, \quad \mathbf{V} = \mathbf{X}\mathbf{W}_V$, with projection matrices $\mathbf{W}_Q, \mathbf{W}_K, \mathbf{W}_V \in \mathbb{R}^{d\times d}$. The rows of $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ represent the query, key, and value vectors, respectively. The core of the mechanism lies in computing relevance scores:

$$\boldsymbol{\alpha}_{i,j} = \frac{\exp\left(\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}\right)}{\sum_{k=1}^{m} \exp\left(\mathbf{q}_i \mathbf{k}_k^\top / \sqrt{d}\right)}$$

where the scaled dot-product $\mathbf{q}_i \mathbf{k}_j^\top / \sqrt{d}$ measures feature compatibility. These normalized coefficients implement a soft reranking mechanism, where the attention coefficients assess the importance of the current feature contribution to `LIKE` decoding. Based on the attention weights, the context matrix $\mathbf{Z} \in \mathbb{R}^{m\times d}$ is computed via value aggregation as follows, where $\mathbf{z}_i$ denotes the i-th row of $\mathbf{Z}$:

$$\mathbf{z}_i = \sum_{j=1}^{m} \boldsymbol{\alpha}_{i,j} \mathbf{v}_j \quad \text{for } i = 1, \ldots, m,$$

This reweighting allows automatic emphasis on logically critical subpatterns (non-wildcard segments) while maintaining long-range dependency awareness - crucial for reconstructing masked segments between wildcards. For example, given a `LIKE` pattern "SIG_O_" to decode, we need to pay more attention to "SIG" than "O" as the prefix "Special Interest Group"("SIG") gives us more details of the underlying matched result than "O". Therefore, these architectural advantages make Transformers naturally suitable for `LIKE` predicate decoding: The dynamic length adaptation handles arbitrary wildcard expansions, while the attention mechanism automatically focuses on discriminative subpatterns to efficiently decode string patterns.

4.3 Capture LIKE pattern in a Seq2Seq manner

SMILE adapts Transformer to perform character-level translation of `LIKE` predicates. As illustrated in Figure 4, the model processes a `LIKE` pattern through distinct encoding and decoding stages.

Encoding Stage: The encoding stage transforms an input `LIKE` pattern $\mathcal{P}$ of length m into a dense vector representation $\mathbf{H} \in \mathbb{R}^{m\times d}$ via three sequential operations: (1) **Character-level Embedding**: Each character in the input pattern is mapped to a d-dimensional vector using a trainable embedding matrix $Emb \in \mathbb{R}^{|V|\times d}$, where V is a column-specific vocabulary. (2) **Positional Encoding**: To inform the model of the character sequence order, which is otherwise ignored by the self-attention mechanism, we inject positional information. This is achieved by summing the character embeddings with a standard sinusoidal positional encoding [99]:

$$\begin{aligned} PE_{(pos,2i)} &= \sin\left(pos \times 10000^{-\frac{2i}{d}}\right), \\ PE_{(pos,2i+1)} &= \cos\left(pos \times 10000^{-\frac{2i}{d}}\right), \end{aligned}$$

where pos is the character's position and i is the dimension index ($i \leq d$). This parameter-free approach facilitates generalization to variable-length patterns, which is crucial for robustly handling

Algorithm 1 Temperature annealed decoding for LIKE patterns

Input: Encoded LIKE pattern $\mathbf{H} \in \mathbb{R}^{m \times d}$, temperature T, beam size K, decode length ℓ, Hash Verifier $Hash$
Output: Valid candidates $C_{\text{valid}} \subseteq C$
1: **procedure** DECODELIKE($\mathbf{H}, T, K, \ell, Hash$)
2: $\quad \mathbf{H}_{\text{tile}} \leftarrow \text{repeat}(\mathbf{H}, K)$ ▷ Tile pattern representation
3: $\quad \mathbf{s_t} \leftarrow [\langle SOS \rangle]^{\times K}$; $\mathbf{done} \leftarrow \text{False}^K$; ▷ Init sequences
4: $\quad$ **for** $t = 1$ **to** ℓ **do**
5: $\quad\quad D_t \leftarrow \text{Decoder}(\mathbf{H}_{\text{tile}}, \mathbf{s_t})$ ▷ Decode
6: $\quad\quad \mathbf{P}_t \leftarrow \frac{\exp(D_t/T)}{\sum_{\epsilon \leq |V|} \exp(D_t[\epsilon]/T)}$ ▷ Temperature scaling
7: $\quad\quad y_t \leftarrow \text{sample}(\mathbf{P}_t)$;
8: $\quad\quad \mathbf{s_t} \leftarrow \mathbf{s_t} + (y_t \cdot \neg \mathbf{done})$; $\mathbf{done} \leftarrow \mathbf{done} \vee (y_t == \langle EOS \rangle)$
9: $\quad\quad$ **if** all(**done**) **then break**
10: $\quad C_{\text{valid}} \leftarrow \{s_t \in \mathbf{s_t} \mid Hash[s_t] = 1\}$ ▷ Hash verification
11: $\quad$ **return** C_{valid}

wildcards. (3) **Self-attention Encoding**: Finally, a Transformer encoder layer processes the resulting sequence. It employs a multi-head self-attention mechanism, followed by layer normalization and a residual connection, to capture long-range dependencies between characters in the pattern.

Decoding stage: The decoder learns a scoring function that maps the encoded LIKE pattern representation $\mathbf{H} \in \mathbb{R}^{m \times d}$ and the currently generated sequence $s_{<t} = (\langle SOS \rangle, x_{\pi 1}, x_{\pi 2}, \ldots, x_{\pi t-1})$ to the vocabulary score distribution D_t of the t-th token. This function is implemented via a Transformer-based architecture, maintaining dimensionality consistency across layers. The decoder then projects these hidden states to the score vector $D_t \in \mathbb{R}^{|V|}$:

$$D_t = \mathbf{W_o} \mathbf{h}_t^N + \mathbf{b_o}$$

where $\mathbf{h}_t^N \in \mathbb{R}^d$ denotes the final layer's hidden representation at step t, $\mathbf{W_o} \in \mathbb{R}^{|V| \times d}$ is the output weight matrix, and $\mathbf{b_o} \in \mathbb{R}^{|V|}$ is the bias vector with $|V|$ being the vocabulary size. The resulting weight vector from the decoder contains the likelihood scores for all tokens in the vocabulary. The subsequent section discusses an efficient decoding strategy based on these scores.

4.4 Translating LIKE Via Temperature Annealed Importance Sampling

Translating a LIKE predicate that matches multiple strings requires generating a diverse set of candidates. Standard greedy decoding, however, often yields repetitive outputs, as it consistently favors high-probability tokens and fails to explore the full solution space [113].

To address this, we employ a temperature-annealed sampling strategy, a technique adapted from large language model decoding [11, 13, 27, 113] to balance exploration and exploitation. Given a LIKE pattern $\mathcal{P}$ over column A with vocabulary V, SMILE's decoder computes a score distribution $D_t \in \mathbb{R}^{|V|}$ at each step t, conditioned on $\mathcal{P}$ and the prefix $s_{<t}$. We then apply temperature scaling to this distribution to promote diversity:

$$\Pr(x_{\pi t} = x_k) = \frac{\exp\left(D_t[k]/T\right)}{\sum_{\epsilon \leq |V|} \exp\left(D_t[\epsilon]/T\right)}$$

Algorithm 1 outlines this process. To generate K candidates concurrently, the encoded LIKE pattern $\mathbf{H}$ is tiled, and each sequence is initialized with a $\langle SOS \rangle$ token (Lines 2-3). The algorithm then iteratively samples tokens. In each step, the decoder generates logits D_t, which are scaled by temperature T to form a probability distribution $\mathbf{P}_t$ (Line 5). A new token y_t is sampled from $\mathbf{P}_t$

and appended to each active sequence (Lines 6-7). The process terminates once all sequences have generated an $\langle EOS \rangle$ token. Finally, a hash-based verifier validates each candidate string in $O(\ell)$ time (Line 10). The overall time complexity is $O(D \cdot \ell)$, where D is the decoder's computational cost and ℓ is the decode length. This complexity is independent of the database size N, ensuring scalability.

4.5 Cardinality-Aware Classification

While SMILE achieves asymptotic acceleration for low-cardinality `LIKE` queries, high-cardinality patterns (e.g., pure `LIKE "%"`) pose challenges. To decode type `LIKE "%"` predicate would require the sampling budget being linear to the table scale. Therefore, such exhaustive generation for non-selective patterns approaches $O(N \times \ell)$ complexity, negating our complexity improvements.

To resolve this tension, we introduce a cardinality-aware classifier that dynamically routes queries between the SLM translator and traditional execution methods. The classifier predicts whether a given `LIKE` pattern will exceed predefined cardinality thresholds proportional to the sampling budget, and we reuse the neural translator's pre-trained encoder (described in §4.3) to obtain $m \times d$ dimensional input patterns' hidden representation $\mathbf{H}$. We then use average pooling to map $\mathbf{H}$ into d dimensional summarization vectors. This summarization captures both syntactic features (wildcard positions, pattern length) and distribution characteristics (expected character distributions). A two-layer MLP with ReLU activation then maps these representations to cardinality class probabilities:

$$\Pr(\text{high-card}) = \sigma(\mathbf{W}_2 \cdot \text{ReLU}(\mathbf{W}_1 \cdot \text{Avg}(\mathbf{H}) + \mathbf{b}_1) + \mathbf{b}_2)$$

where $\text{Avg}(\mathbf{H})$ denotes average pooling of the input `LIKE` pattern's encoded representation $\mathbf{H}$, and σ represents the sigmoid function. The model trains on query logs annotated with true cardinality values, optimizing binary cross-entropy loss. In practice, this lightweight classifier (6MB parameters) introduces negligible overhead (0.1ms/query) while achieving accurate query routing in our experiments, enabling reliable fallback to full scans for low-selectivity patterns without compromising low-latency acceleration for targeted queries.

5 Guarantees and Extensions

Having established the architecture of SMILE, we now turn to its theoretical underpinnings and practical extensibility. This section first develops a framework for probabilistic approximation guarantees, formally quantifying the relationship between the sampling effort of our SLM-based translator and the completeness of the query result. Building upon this theoretical foundation, we then explore practical extensions of our methodology.

5.1 Probabilistic Guarantees for Approximation

To develop a principled understanding of our language model based approach for `LIKE` predicate approximation, we construct a theoretical model. The goal of this model is to quantify the relationship between a query's complexity, the model's capacity, and the number of samples, n, required to retrieve at least K matching strings with confidence $1 - \delta$.

Our framework's central premise is to contrast the language model's objective with the conventional `LIKE` execution, which is a *discriminative* process: given a fixed, finite set of stored strings, the system identifies which ones satisfy the `LIKE` pattern. In contrast, our language model tackles this a *generative* task: it must infer the syntactic constraints implicit in the pattern and synthesize valid strings *de novo*. Hence, the difficulty of this task depends not merely on the empirical data distribution of any specific database instance, but more on the intrinsic structural complexity of the pattern itself.

Assumption 1 (Task Complexity as Generative Entropy). We model the complexity of a `LIKE` pattern $\mathcal{P}$ by the difficulty of the generative task it represents. This difficulty is proportional to the size of its corresponding language, $L(\mathcal{P})$—the space of all possible valid strings. To formally quantify this *intrinsic generative complexity* while abstracting away from the contingent data distribution of any particular database, we employ the principle of maximum entropy. This leads us to assume a uniform distribution over all syntactically valid strings in $L(\mathcal{P})$, thereby isolating the complexity inherent in the pattern's structure.

Definition 1 (Pattern Entropy). Under this generative framework, for a LIKE pattern $\mathcal{P}$ over an alphabet Σ, we define the **pattern entropy**, $H(\mathcal{P})$, as the Shannon entropy of the uniform distribution over its language $L(\mathcal{P})$, measured in nats:

$$H(\mathcal{P}) = \ln |L(\mathcal{P})| \tag{1}$$

This value serves as a proxy for the syntactic learning difficulty posed to the generative model.

Theorem 5.1 (Entropy of a LIKE Pattern). *Given a* LIKE *pattern of length x with p underscore ("_") and q percent ("%") wildcards over alphabet Σ, no two percent signs adjacent, and a maximum generated-string length of m, the pattern entropy $H(\mathcal{P})$ is:*

$$H(\mathcal{P}) = p \ln |\Sigma| + \ln \left(\sum_{s=0}^{m-x+q} \binom{s+q-1}{q-1} |\Sigma|^s \right) \tag{2}$$

The proof is detailed in Appendix I.1. Theorem 5.1 provides a closed-form method to compute this complexity measure. The total entropy of a workload $\mathbb{W}$ is the sum of the entropies of its constituent patterns: $H(\mathbb{W}) = \sum_{\mathcal{P} \in \mathbb{W}} H(\mathcal{P})$. Next, we model the language model's capabilities in a simplified manner.

Assumption 2 (Model Information Capacity). We followed Meta AI's recent analyses [74], postulate that a model with $|\theta|$ parameters has an effective information capacity, M, that is proportional to its size, in a linear relationship:

$$M = \alpha |\theta| \tag{3}$$

where α is an architecture-dependent constant. Within our framework, M represents an abstracted upper bound on the total task complexity $H(\mathbb{W})$ that the model can reliably internalize.

Assumption 3 (Capacity-Limited Performance). We postulate a relationship between the model's capacity and its generative success probability, p. We conceptualize this as a resource allocation problem. If the model's capacity M meets or exceeds the task's complexity $H(\mathbb{W})$, its performance is primarily limited by other factors (e.g., training quality, decoding stochasticity), which we bound with a universal constant $c \in (0, 1]$. If the model is under-capacitated ($M < H(\mathbb{W})$), we assume its performance degrades proportionally to the ratio of available capacity to task demand. This leads to the following lower bound on the success probability:

$$p \geq c \cdot \min\left(1, \frac{M}{H(\mathbb{W})}\right) = c \cdot \min\left(1, \frac{\alpha|\theta|}{H(\mathbb{W})}\right) \tag{4}$$

This assumption provides a tractable, albeit simplified, link between model scale and task complexity, enabling the derivation of sampling bounds.

Theorem 5.2 (Sampling Complexity for LIKE Approximation). *Under the assumptions of our theoretical model, for a query workload with entropy $H(\mathbb{W})$, to retrieve at least K correct results with confidence $1 - \delta$, the required number of samples n from a model with $|\theta|$ parameters must satisfy:*

$$n \geq \begin{cases} \frac{1}{c}\left(K + \ln(1/\delta) + \sqrt{(2K + \ln(1/\delta))\ln(1/\delta)}\right) & \text{if } H(\mathbb{W}) \leq \alpha|\theta| \\ \frac{H(\mathbb{W})}{c \cdot \alpha|\theta|}\left(K + \ln(1/\delta) + \sqrt{(2K + \ln(1/\delta))\ln(1/\delta)}\right) & \text{otherwise} \end{cases} \tag{5}$$

The proof is detailed in Appendix I.2. Theorem 5.2 is the central result of our model, establishing a quantitative link between a workload's intrinsic complexity ($H(\mathbb{W})$), the model's abstracted capacity ($|\theta|, \alpha$), and the sampling budget (n) required for a given performance target (K, δ). Crucially, the generative viewpoint of our model implies that for a consistent query distribution, the syntactic rules learned from one data sample can generalize to entirely disjoint data partitions. This property is instrumental for handling evolving databases and enables rapid model deployment, a claim we validate empirically in Section 6.3 and 6.4.

5.2 Discussions

We discuss SMILE's extensibility to long text columns and its broader applications.

Scalability to Long Text Columns. Deep sequence models for `LIKE` optimization struggle with long text columns [7, 59, 88],[1] a limitation also present in our baseline full-string generation approach. The core challenges are twofold: (1) long text implies longer m in Theorem 5.1, thus greater entropy, impedes learning the data distribution for exact matches, and (2) the linear cost of autoregressive inference becomes prohibitive. To address this, we propose a hybrid strategy that reframes SMILE as a prefix-guided query rewriter. Instead of generating a full string, the SLM produces a fixed-length, high-selectivity prefix. This prefix is then used to probe a standard index (e.g., a B^+-tree), converting a `LIKE "%keyword%"` query into an efficient `LIKE "prefix%"` lookup. This design solves both issues: it lowers the target entropy ($H(x_{\text{prefix}}) < H(x_{\text{full}})$), improving sampling success per Theorem 5.2, and decouples inference cost from string length to ensure scalability. We believe this is the first practical deep learning approach for `LIKE` acceleration that scales to the maximum wildcard lengths (e.g., 8KB) supported by modern database systems.[2]

Generalization to Other Tasks. SMILE's core methodology is broadly applicable to other tasks. For instance, SMILE can accelerate approximate regular expression (regex) matching. By modeling regex as a sequence-to-sequence task, it can significantly improve performance in interactive data exploration and cleaning workflows. The approach also extends to optimizing fuzzy string joins (e.g., "Orders.CustomerName" with "Customers.Name"). A trained SLM can act as a candidate generator, mapping an input like "J. Smith" to high-probability matches such as "John Smith". This transforms an expensive fuzzy join into a series of efficient, model-guided lookups. While primarily designed for single-column approximate queries, SMILE can be readily extended to optimize complex analytical queries, serving as a specialized form of Approximate Query Processing [15, 76]. We employ SMILE as a query rewriter for queries with non-sargable "LIKE" predicates: it generates a candidate set of matching strings and transforms the predicate into an equivalent "WHERE column IN (...)" clause. This simple yet powerful rewrite makes the predicate index-friendly, enabling the database optimizer to leverage efficient hash-based lookups instead of costly scans. If the cardinality estimator in the database predicts high selectivity for this predicate, SMILE can be enabled to perform the above rewriting. We validated this strategy end-to-end on relevant queries from TPC-H and IMDB-JOB benchmarks in Sec 6.2 and Appendix H.

6 EXPERIMENTS

In this section, we provide an experimental analysis of SMILE in accelerating approximated `LIKE` pattern. We use our experiment results to answer the following questions:

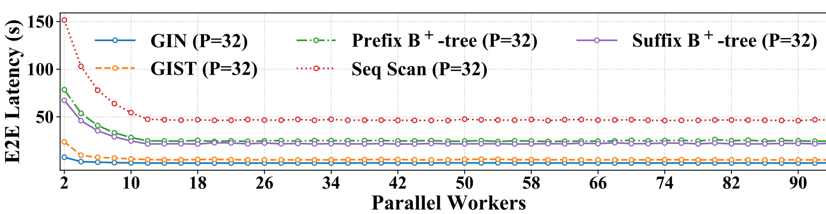
(Q1) Effectiveness: Can the proposed language-model-based `LIKE` predicate decoding strategy significantly reduce the execution time and cost of `LIKE` predicates compared to traditional methods and state-of-the-art large language models across multiple datasets? Can it generate sufficiently

[1]Recent deep LIKE CardEst works use strings of 50–100 length in average, none exceeding 500. [7, 59]

[2]For example, 255 bytes in SAP [85], 4KB in DB2 [47], and 8KB in SQL Server [71].

high-quality candidate sets to recall potential query results? Can it play a useful optimization role in practical complex workloads? (Section 6.2)
(Q2) Scalability: Is the method's scalability in training and inference time sufficiently effective? Is the method's space occupancy sufficiently low? How does SMILE's performance and accuracy scale with increasing workload entropy and string length, particularly on billion-record and long-text datasets?(Section 6.3)
(Q3) Robustness: If data distribution drifts occur, can the method be updated in time to adapt to data changes? If the query template distribution changes, can the method still effectively support unknown query template estimation without retraining? (Section 6.4)

6.1 Experimental Setup



(a) Baseline Configuration Tuning(Partition=32).
(b) Hardware Hourly Rental Costs.

Hardware Configuration	Hourly Rental Cost ($)
1 × AMD EPYC 9654 CPU (96 Cores) [21]	$1.50
1 × AMD EPYC 9654 CPU (16 Cores) [21]	$0.25
1 × NVIDIA H20 GPU + 1 × AMD EPYC 9654 CPU (16 Cores) [6]	$0.98

Fig. 5. Configuration analysis and hardware cost.

Datasets: We evaluate `LIKE` predicate performance on six diverse datasets, with detailed statistics summarized in Table 1. Four are standard benchmarks in string query research [7, 59, 107]: (1) **TPC-H** [97], comprising 24 million rows from the `comment` column of the `lineitem` table (SF=4); (2) **IMDB** [48], with 15 million entries from the `primaryName` column of the `Name` table of the **real-world** IMDB dataset; (3) **Reddit** [73], containing 2 million **real-world** reddit usernames; and (4) **WIKI** [101], consisting of 4 million **real-world** Wikipedia article titles. To further assess scalability and robustness, we use two large-scale, **real-world** datasets: (5) **RedPajama** [95], a set of 1 billion unique string windows (max length 40) extracted from a 1TB corpus; and (6) **News Room** [44], used for long-text evaluation with 1.2 million articles averaging 3962 characters in length.

Table 1. Statistics for the datasets used in our evaluation.

	Reddit	WIKI	IMDB	TPC-H	Redpajama	NewsRoom
Total	2,000,000	4,000,000	14,757,614	23,996,604	1,000,000,000	1,212,741
Unique	1,999,847	4,000,000	11,278,588	15,189,929	1,000,000,000	1,200,762
Min Length	1	1	1	8	1	27
Max Length	20	257	105	43	40	18386
Avg Length	11.00	19.91	13.51	26.23	36.61	3962.47

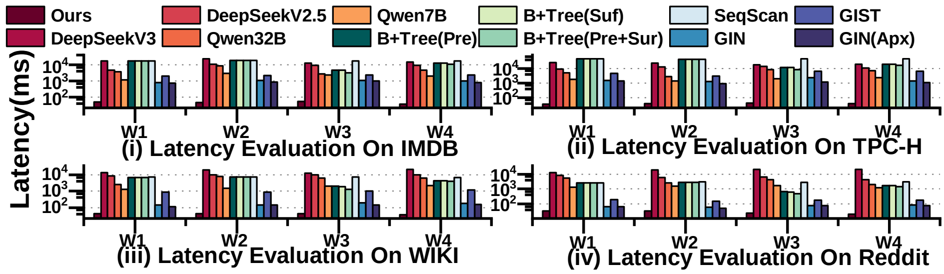
Workload Design. To evaluate the robustness of our method, we designed four `LIKE` workloads (W1-W4) that span from realistic to challenging scenarios. **Realism-Driven Workloads (W1 & W2).** We derived W1 and W2 from an analysis in Appendix A based on our collected 84,047 production `LIKE` queries(available at [4]). The analysis revealed that challenging infix searches (e.g., `%Keyword%`, `%Key1%Key2%`) are prevalent. We found that in such queries, the substrings masked by wildcards often correspond to linguistic units like morphemes or syllables, rather than arbitrary characters. Accordingly, we generate realistic patterns by splitting string into list of linguistic units using [2, 3] and replacing 0-2 consecutive linguistic units with a single "%" wildcard. This method covers over 50% of the production cases (Appendix A.3). We therefore define two specific workloads: **W1 (Syllable-based):** Simulates phonetic recall by masking syllables, the basic units of pronunciation [83]. For example, `"SIGMOD CONFERENCE"` can yield the pattern `%MOD CON%EN%`. **W2 (Morpheme-based):** Models semantic-driven searches by masking morphemes, the smallest meaningful units in a language [32, 40]. This simulates users searching for concepts embedded within larger words. The detailed construction algorithm of these workload is described in Appendix E. **Benchmark Workloads (W3 & W4).** To benchmark against established standards and test performance boundaries, we include two additional workloads: **W3 (CLIQUE Workload):** Adopts the standard suite of `LIKE` patterns from CLIQUE [59] (e.g., `%Key`, `Key%`, `%Key%Key%`) to evaluate performance on common predicate forms. **W4 (LPLM Workload):** Uses the generator from LPLM [7] to create challenging patterns with high wildcard density and low semantic content (e.g., `S0%S1%S2%...Sn%`).

Finally, our analysis in Appendix A.4 and A.6 shows that single-character "_" wildcards are statistically independent of "%" wildcards, and usually appear 0-5 times per query, with occurrences of 6-7 times being less than 0.1%. Based on this finding, we augment all generated patterns across W1-W4 by randomly replacing 0-5 characters in each key with a "_", enhancing the realism of the final query predicates.

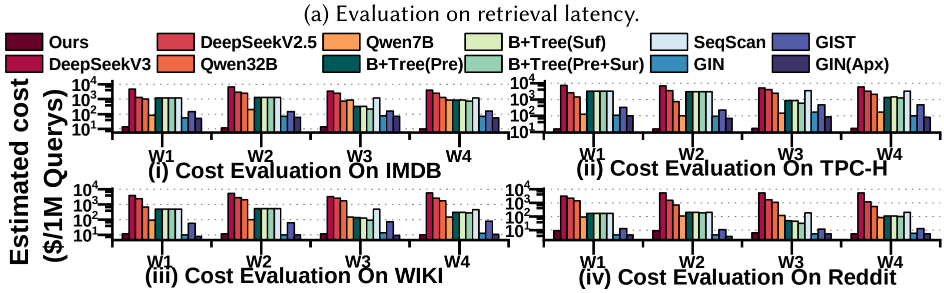
Baseline competitors: We select the following baseline competitors as they have proven their effectiveness in realistic database `LIKE` pattern matching deployments of PostgreSQL, or achieved state-of-the-art open-sourced text pattern understanding performance in relevant NLP tasks [24, 93, 94]. The baselines include:

1. *Prefix B+-tree*: Accelerates prefix `LIKE` patterns through prefix-optimized B+-tree structure.
2. *Suffix B+-tree*: Accelerates suffix `LIKE` patterns using reverse-ordered B+-tree structure.
3. *Prefix+Suffix B+-tree*: Accelerates suffix `LIKE` patterns using combined prefix and suffix B+-trees.
4. *Sequential scan*: Baseline full-table scanning approach.
5. *GIST index* [29]: Accelerates `LIKE` via Generalized Search Tree.
6. *GIN index* [77]: Accelerates `LIKE` through Generalized Inverted Index.
7. *GIN index (Approximate)* [29]: Approximate `LIKE` matching using inverted index, following a similarity filtering technique in [16, 43].
8. *Qwen2.5-7B* [94]: 7B-parameter LLM developed by Qwen-Team.
9. *Qwen2.5-32B* [94]: 32B-parameter LLM developed by Qwen-Team.
10. *DeepSeek-V2.5* [24]: 236B-parameter LLM developed by Deepseek.
11. *DeepSeek-V3* [93]: State-of-the-art open-sourced 671B LLM.

Experimental settings: The language model of SMILE is configured with a hidden layer size of 512 and a single attention layer. During training, we set the learning rate to 3×10^{-4} and the batch size to 16384. For the SMILE's classifier, we maintain the same hidden layer size of 512 but adjust the batch size to 2048 and the learning rate to 1×10^{-4}. We establish a classification threshold of 16 for the query workload cardinality. When training the small language model, we utilize 200 sampled batches to ensure effective learning and model convergence. During inference, we set the sampling number being 64. Our SMILE is trained and evaluated on a server with a Nvidia H20 GPU and a 96-core AMD EPYC 9654 CPU. The PostgreSQL version is 16.3. To ensure a fair comparison, we first tune the optimal configuration for baselines via W3 of the TPC-H dataset on the 96-core CPU. As shown in Figure 5(a), their performance saturates at 16 parallel workers due to Amdahl's Law [5]. Beyond this point, more cores offer no advantage. Consequently, we configured all CPU baselines with 16 workers and 32 partitions for the main experiments. Our 96-core server's resources can be



(a) Evaluation on retrieval latency.



(b) Evaluation on uniformed retrieval cost

Fig. 6. Evaluation on retrieval efficiency (Y-axis in log scale).

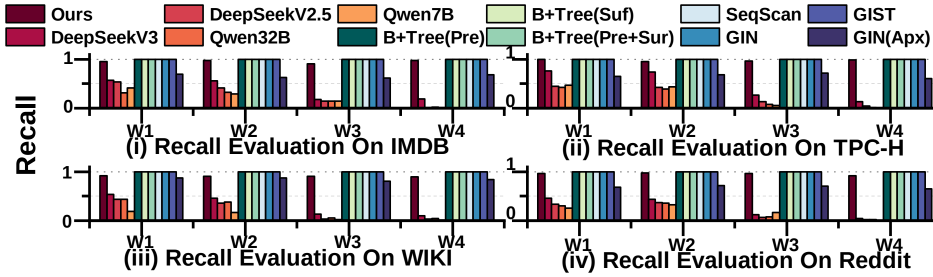
partitioned, simulating a multi-tenant environment. Therefore, to provide a practical and equitable comparison of economic efficiency, our cost analysis for all methods is normalized to the price of a 16-core CPU setting, which represents the most cost-effective hardware configuration. This methodology ensures a fair comparison of economic efficiency (see Figure 5(b) and Appendix G).

Evaluation metrics: We assess method effectiveness along three principal dimensions. **End-to-end retrieval latency**: Total query processing time. **Recall**: Result quality, defined as Recall = $\frac{|R_1 \cap R_2|}{|R_1|}$, where R_1 denotes the ground-truth result set and R_2 the retrieved set. **Uniformed-Cost**: To enable hardware-agnostic comparison, we adopt the rental-cost normalization from [87]. Results are reported as estimated cost per one million queries ($/1M queries), using the hourly hardware rental rates in Fig. 5(b); our full cost model is detailed in Appendix G.

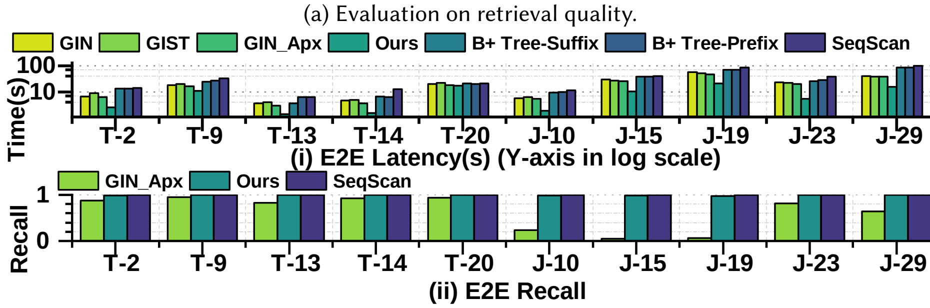
6.2 Effectiveness Evaluation

In this section, we evaluate the effectiveness of multiple LIKE retrieval methods across four datasets under four different workload patterns. We assess the effectiveness and quality of these methods from two perspectives: efficiency and quality.

Efficiency Evaluation: We conducted a comprehensive efficiency evaluation on four diverse datasets (IMDB, TPC-H, WIKI, and Reddit), assessing performance through two key metrics: raw retrieval latency and a normalized query processing cost. As illustrated in Figure 6(a), SMILE consistently demonstrates superior retrieval latency. It achieves a 1.88−41.56× speedup over trigram-based GIN indexes and surpasses both prefix/suffix B+-trees and full-table scans by two and three orders of magnitude, respectively. This performance stems from SMILE's design, which synergizes $O(\ell)$-time decoding with a lightweight, hash-based candidate validation mechanism. This approach effectively circumvents the inherent bottlenecks of traditional methods, such as the $O(\log N)$ complexity of tree-based indexes or the performance degradation of GIN when wildcards disrupt trigram extraction. To ensure a fair comparison, particularly given SMILE's use of GPU hardware, we also introduce a normalized cost metric (USD per million queries) that accounts for hardware and energy expenditure. This provides an equitable assessment against CPU-based baselines, with



(a) Evaluation on retrieval quality.



(b) End-to-end evaluation on IMDB-JOB(J) and TPC-H(T) templates.

Fig. 7. Evaluation on retrieval quality and end-to-end (E2E) performance.

results shown in Figure 6(b). On large-scale, complex datasets like IMDB and TPC-H, SMILE's efficiency translates into a substantial cost advantage, proving to be one to two orders of magnitude more cost-effective than all baselines. For instance, its cost on IMDB is significantly lower than that of the GIN index. However, on smaller datasets like Reddit, where the overhead of our model is less amortized, the cost advantage narrows. Under such smaller dataset, the highly-optimized GIN index is 2× more cost effective than SMILE. In summary, while GIN remains a viable option for smaller workloads, SMILE's architecture delivers both superior speed and unparalleled cost-efficiency for the large-scale, complex pattern matching tasks that are the primary target of modern data systems.

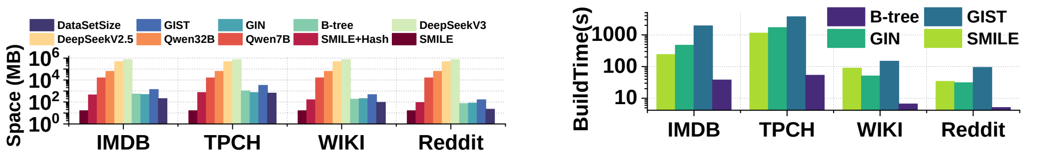
Quality evaluation: Figure 7(a) evaluates `LIKE` search quality across four datasets and query patterns. While conventional deterministic indexes ensure perfect recall at a high computational cost, approximate methods like `GIN(Apx)` falter on most datasets. LLM-based approaches show performance scaling with model size; large models like `DeepSeek-V3` (671B) achieve certain recall up to 0.8 on simple patterns (W1, W2), but their performance collapses (<0.1) on complex ones (W3, W4), revealing an inability to grasp data semantics in a database specific domain. In stark contrast, our compact `SMILE` model consistently delivers high recall (>0.9) across all workloads. This demonstrates the efficacy of a specialized, data-aware model for `LIKE` predicate evaluation, positioning `SMILE` as a robust solution where both accuracy and efficiency are critical.

End-to-End Performance on Realistic Workloads. Figure 7(b) presents a comprehensive end-to-end (E2E) evaluation of SMILE on complex queries from the TPC-H (T) and IMDB-JOB (J) benchmarks, selected for their challenging `LIKE` predicates. As shown in Figure 7(b)(i), SMILE consistently outperforms all baselines, achieving speedups of 1.18× to 9.51× over the next-best method. This demonstrates its practical utility in accelerating full query plans that include complex operations like joins and aggregations. Figure 7(b)(ii) confirms that this performance gain is achieved while maintaining a consistently high recall of over 95%. This evaluation also highlights the structural fragility of trigram-based approximate indexes. Notably, `GIN(Apx)` exhibits highly inconsistent recall—near-perfect on TPC-H queries but collapsing to near-zero on several IMDB-JOB templates. This discrepancy stems from the query structure: TPC-H queries often terminate with a GROUP BY operation, which can mask row-level filtering errors in the final aggregated output. Conversely, the IMDB-JOB queries are direct selections where any filtering inaccuracy immediately impacts the final result set. SMILE's stable, high recall across both benchmarks underscores its robustness. While the substantial speedup from accelerating the LIKE predicate is naturally diluted by other system costs (e.g., I/O, joins) in E2E queries, the fundamental operator acceleration provides significant practical value. A more granular analysis is available in Appendix H.

Takeaway: SMILE delivers both high efficiency and high recall for LIKE retrieval across diverse workloads. *Efficiency-wise*, it outperforms trigram indexes, B^+-trees, and full scans, thanks to $O(\ell)$ decoding and lightweight hash-based validation—avoiding the bottlenecks of tree traversal, index degeneration, and LLM overhead. *Quality-wise*, SMILE achieves over 0.9 recall on all patterns, far surpassing LLMs while matching deterministic indexes. Its compact design enables sub-20 ms inference without sacrificing accuracy, making it practical for accelerating complex queries.

6.3 Scalability Evaluation

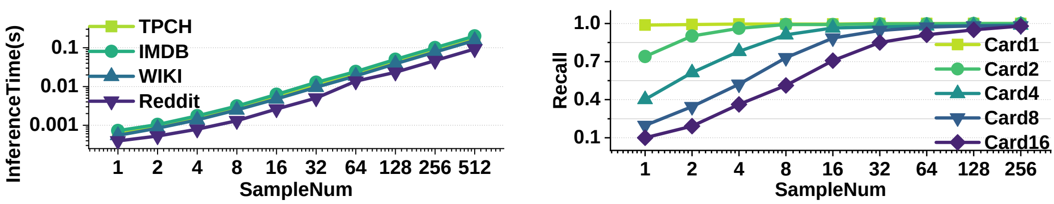
In this section, we empirically investigate the scalability of our approach through experiments, analyzing storage overhead, training cost, and inference cost.
Space scalability: Figure 8a evaluates the storage overhead of SMILE against state-of-the-art LLMs and traditional LIKE-optimized indexes. SMILE requires only ≈16MB of parameter storage, which is 3–5 orders of magnitude less than large LLMs (e.g., DeepSeek-V3, Qwen-32B). Compared to GIN, GIST, and B+-trees, SMILE reduces space consumption by 23×, 82×, and 28× in average. This efficiency arises from two factors: (1) our model's use of attention mechanisms to compactly encode workload distribution, and (2) its specialized design for database predicates, avoiding the domain-agnostic parameter bloat of general-purpose LLMs. These results confirm that SMILE achieves exceptional space efficiency without compromising task-specific effectiveness.
Training scalability: Fig. 8b evaluates SMILE's training-phase scalability against traditional indexes. On larger datasets (IMDB, TPC-H), SMILE's construction time is comparable to GIN and five times faster than GIST, though B+-tree remains fastest due to its batched, sorted-data indexing. GIN and GIST incur overhead from tokenizing column data into triplets and building inverted/general search trees over expanded token sets. SMILE exploits GPU parallelism via its attention mechanism, enabling concurrent gradient computation and updates for 10^4 tuple pairs per batch. Its compact parameter count further reduces convergence time. Consequently, SMILE achieves superior training scalability through hardware parallelism and attention efficiency.
Inference scalability: Fig. 9a shows end-to-end latency under varying sample budgets. Latency scales linearly with sample count: below 1ms for 8 samples, and close to 100ms for 512. At maximum sampling (512), this 0.1s latency is competitive with existing baselines under equivalent workloads, demonstrating superior throughput for high-volume inference.
Scalability with Cardinality. Figure 9b evaluates SMILE's decoding scalability with respect to query cardinality on the TPC-H lineitem table. A fixed sample budget often suffices to achieve high recall (≥ 90%), confirming SMILE's efficiency. Moreover, the sample size needed to maintain high recall scales approximately linearly with result cardinality. This aligns with Theorem 5.2: higher-cardinality queries typically exhibit greater entropy and complexity, and the theorem predicts that the required sample size must scale accordingly—empirically observed here as a linear dependence on result size, thereby validating our theoretical bound.
Mitigating High-Entropy Challenges. High-entropy queries are inherently difficult, but our theory (Eq. 4) suggests a remedy: increasing model capacity, $|\theta|$. To validate this, we study the interplay among model size, query entropy, and recall. Figure 9c shows a representative result



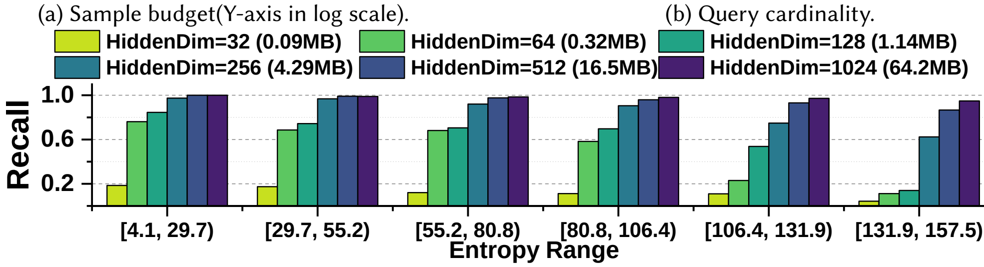
(a) Scalability on space. (b) Scalability on building time.

Fig. 8. Scalability on space and building time(Y-axis in log scale).

(Appendix C provide details). Entropy is calcualted using Eq. 2. Two insights emerge: (1) for a fixed model size, recall degrades as entropy increases, confirming entropy as a performance boundary, as predicted; and (2) this boundary is malleable—increasing model capacity significantly boosts recall, especially in high-entropy regimes. This reveals a practical scaling law: the performance penalty from high entropy can be effectively offset by larger models, offering a principled trade-off between capacity and robustness on complex workloads. See Appendix C for a comprehensive analysis.



(a) Sample budget(Y-axis in log scale). (b) Query cardinality.



(c) Impact of model capacity across different query entropy regimes.

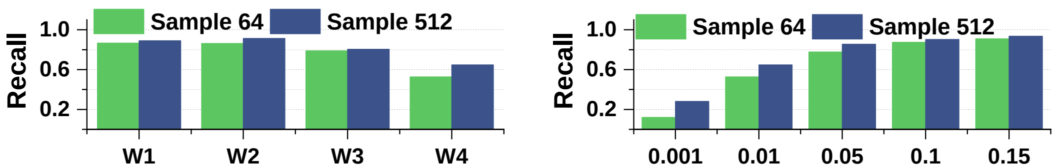
Fig. 9. Scalability on budget, cardinality, entropy.

(a) SMILE (16MB) on TPC-H. (b) GPT-2 (124M) on NewsRoom.

Fig. 10. Scalability with respect to string length.

Scalability with String Length. Figure 10 evaluates the scalability of our approach with respect to the length of the LIKE predicate. Figure 10a shows the performance of our standard 16MB SMILE model on the IMDB dataset for short predicates (up to 45 characters). As expected, latency scales linearly with predicate length, while recall remains consistently high. Figure 10b) demonstrates our system's ability to handle extremely long predicates, a significant challenge for traditional DBMS [71, 85]. This experiment, conducted on W4 of the News Room dataset, utilizes an extension of our prefix generation technique (Section 5.2) with a fine-tuned 124M GPT-2 model. As detailed in Appendix B.2, we find that a remarkably small number of generated prefix samples is sufficient for high accuracy. With just 8 samples, we achieve recall above 0.95 for predicates up to 8192 characters.

This result is achieved with modest overhead: latency remains under 100ms, showcasing a highly favorable trade-off between accuracy and performance. A comprehensive analysis of the interplay between sample count, latency, and recall is provided in Appendix B.2.



(a) Generalization from 1% data. (b) Varying data proportion(W4).

Fig. 11. Scalability on the billion-scale RedPajama dataset.

Scalability Across Billion-Scale Data. To evaluate the scalability and cost-effectiveness of our approach, we conducted experiments on the billion-record RedPajama dataset. First, we trained a 16MB SMILE model on 1% of the data and tested its generalization on an unseen, disjoint subset. As shown in Figure 11a, the model effectively learned the data distribution for simpler workloads, achieving over 0.9 recall for W2 with an inference sample size of 512. However, performance on the more complex workload W4 was lower, indicating the need for more training data. To determine the sufficient data scale, we trained models on progressively larger fractions of the dataset, focusing on the challenging W4 workload. Figure 11b shows that recall improves significantly as the training data proportion increases from 0.1% to 10%. Critically, we found that training on 10% of the data is sufficient to achieve excellent performance, yielding 0.9 recall with 512 samples. These results demonstrate that our method is both scalable and data-efficient. This finding corroborates that high performance does not require the full dataset or oversized models. Since query workload distributions are similar, the model can learn a generalizable mapping from a representative subset, making performance largely independent of absolute data scale once sufficiency is reached.

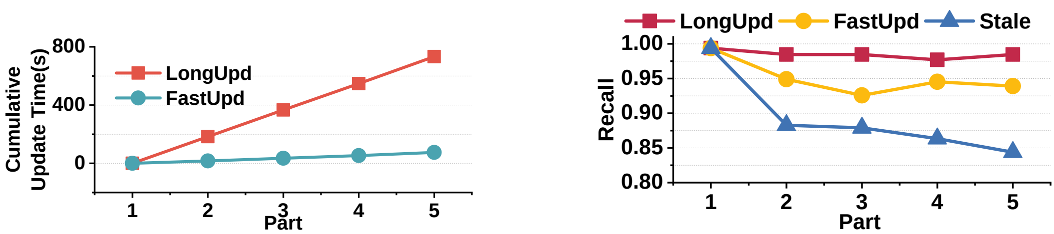
Takeaway: Our evaluation confirms SMILE is a practical and resource-efficient solution for LIKE queries, demonstrating superior scalability: **1. On Space:** 3–5 orders of magnitude smaller than LLMs. **2. On Time:** Achieves competitive build times and comparable inference latency to traditional methods, while scaling gracefully to long predicates. **3. On Data Efficiency:** Generalizes on billion-scale data by training on only a small fraction (e.g., 10%), validating its efficiency.

6.4 Robustness Evaluation

The effectiveness of SMILE primarily derives from two key factors: the learned joint character distribution across database columns and the query patterns extracted from historical workloads. To systematically evaluate the robustness of SMILE under deviations from normal conditions, we conduct controlled experiments simulating both query drift and data drift scenarios using the realworld IMDB dataset (15M tuples with maximum string length attributes). This dataset presents significant robustness challenges due to its complex string patterns and scale.

Robustness to Data Drifts. We evaluate SMILE's robustness to data drifts by simulating periodic data ingestion. The IMDB-name table is partitioned into five segments, and we test three variants: *Stale* (no updates), *FastUpd* (fine-tuned on 10 batches from each new partition), and *LongUpd* (fine-tuned on 100 batches).

As shown in Fig. 12, the results highlight an effective trade-off between performance and update cost. Remarkably, the *Stale* model maintains 84% recall on the final partition, demonstrating strong generalization from its initial training. The *FastUpd* strategy offers an efficient compromise, restoring recall close to 95% with a lightweight fine-tuning process that totals only 75 seconds. For maximum precision, *LongUpd* achieves near-perfect recall at a higher computational cost. This experiment



(a) Update Scalability. (b) Recall Robustness.

Fig. 12. Performance of SMILE under data drifts, showing the trade-off between update cost and recall.

confirms three key properties of SMILE: **(P1) Inherent Robustness:** It naturally generalizes to data with similar distributions, even without updates. **(P2) Efficient Adaptation:** Lightweight fine-tuning provides a cost-effective method to maintain high accuracy on evolving data. **(P3) Flexible Precision:** The architecture allows for more intensive retraining to achieve maximum precision when resources permit. These validate SMILE's suitability for dynamic environments.

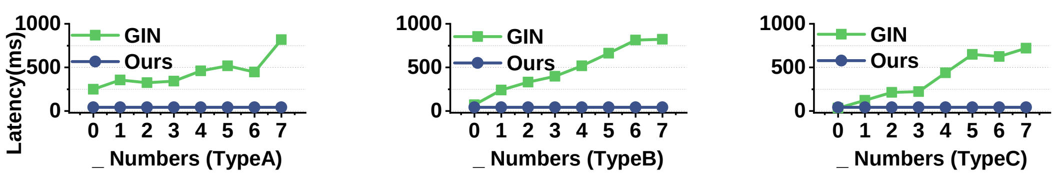
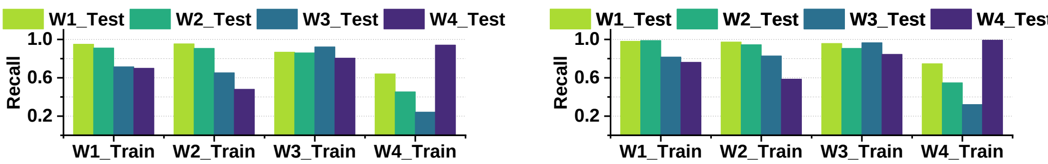


Fig. 13. Robustness on wildcard patterns against inverted index.

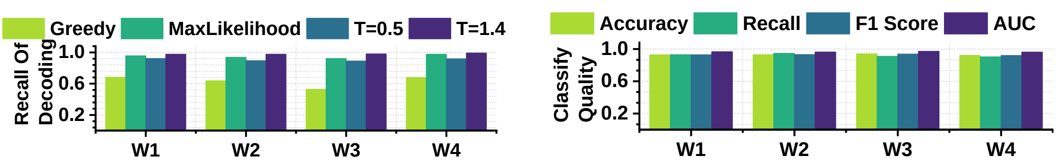
Robustness on patterns. To evaluate robustness, we compare our method against inverted index across three workloads shown in Figure 13. The workloads are designed around keyword selectivity, where a keyword is defined as high-selectivity if its data frequency is below 0.5% and low-selectivity if above 2%. **Type-A** queries ("%low%low%") conjoin two common keywords. **Type-B** ("%high%low%") contains at least one rare keyword. **Type-C** ("%word%") contains single keyword with equal mix of high and low selectivity keywords. To simulate real-world complexity, we inject a varying number of "_" wildcards (0-7) into each pattern. The results highlight a clear performance trade-off. For Type-A queries, our generative approach is substantially faster, as it holistically processes the pattern, bypassing the GIN index's expensive intersection of large posting lists. On Type-B and Type-C queries, performance is comparable since GIN can leverage a highly selective keyword or perform a direct lookup. Critically, Figure 13 shows that GIN's latency degrades significantly with more "_" wildcards, as they fragment the trigrams the index relies on. In contrast, our method's performance remains stable, demonstrating superior robustness to the complex and fragmented patterns that challenge traditional n-gram indexes. See Appendix D for a more analysis.
Robustness to Query Drifts. In Fig. 14, we investigate SMILE's OOD generalization by cross-evaluating on workloads with varying structural characteristics (W1-W4). Our analysis reveals two primary findings. First, a model trained on the diverse patterns of the W3 workload acts as a robust "generalist," adapting effectively to the realistic query structures of W1 and W2. This indicates that SMILE learns fundamental wildcard logic, not just template-specific features. However, generalization proves challenging when faced with the highly arbitrary and hard patterns of the W4 workload, which represents a significant structural shift and defines the model's performance boundaries. Second, and more importantly, this generalization gap can be substantially mitigated by increasing the inference-time sampling budget. As shown by comparing Fig. 14a and Fig. 14b, a larger sampling scale boosts recall by 10-20% on OOD templates. This highlights a key practical

advantage of our approach: the ability to trade computational resources for enhanced robustness against unforeseen query drifts, effectively adapting to new patterns without retraining [25, 90, 93].



(a) Query OOD eval (S-32). (b) Query OOD eval (S-256).

Fig. 14. Query drift evaluation.



(a) Decoding robustness. (b) Classification quality.

Fig. 15. Decoding/Classification robustness.

Robustness of Different Decoding Methods. Fig. 15a evaluates the robustness of four decoding strategies for SMILE trained on templates from W1–W4 and tested on a disjoint set from the same wildcard workload distribution. Greedy decoding demonstrates poor generalization, frequently generating repetitive outputs and failing. Maximum likelihood and low-temperature ($T = 0.5$) decoding provide only slight improvements and remain ineffective on the more demanding W3 and W4 workloads. Conversely, high-temperature ($T = 1.4$) decoding achieves an optimal trade-off between quality and diversity, maintaining high recall across all workload patterns.

Robustness of the cardinality classifier: Fig. 15b assesses the classifier's ability to identify high-cardinality queries (threshold: 16) under varying wildcard patterns. Four classifier variants, each tuned to a specific workload pattern, achieve F1-scores >90% across all configurations. This performance arises from the SLM encoder's semantic representation, which maps query patterns into a latent space where high-cardinality queries become linearly separable. The resulting features enable the neural classifier to reliably identify and route such queries, enhancing processing efficiency.

> **Takeaway:** SMILE exhibits strong robustness to both data and query drift. It generalizes effectively to evolving data, enabling a flexible trade-off between accuracy and fine-tuning cost. Unlike traditional inverted indexes, its performance remains stable across complex wildcard patterns. Meanwhile, SMILE mitigates query drift by trading inference-time computation for generalization: increasing the sampling budget significantly improves zero-shot recall on unseen query templates, offering a practical alternative to expensive retraining.

7 Related Work

Learned cardinality estimation on `LIKE` predicates: Cardinality estimation refers to the problem of predicting the result size of a query without executing it [45], leveraging either data distributions [67, 108, 109] or query patterns [54, 66]. Recently, learned approaches have become prominent in this domain, offering superior accuracy compared to traditional histogram or sampling-based methods [109]. In terms of string `LIKE` predicates, many works target estimating the cardinality of such predicates using learned models [7, 50, 59, 60]. However, there exists a fundamental difference between the cardinality estimation problem studied in prior work and our focus on accelerating `LIKE` predicates. While cardinality estimation aims to predict the number of matching tuples, `LIKE` predicate acceleration focuses on optimizing the end-to-end execution efficiency of queries involving LIKE operations. Prior methods addressing LIKE cardinality estimation do not address the core challenge of reducing the computational overhead inherent in pattern-matching operations, which is the central objective of our proposed solution.

Learned query acceleration: Recent research efforts increasingly replace traditional data structures with machine learning components [28, 41, 56, 57, 84, 98, 102]. Learned indexes revolutionize query processing by substituting logarithmic-time tree traversals with constant-time ML model inference [41, 56, 102]. For relational data, the Recursive Model Index (RMI) [56] employs multi-layer perceptrons to predict record locations, while the Fitting-Tree [41] achieves error-bounded performance through piecewise linear regression. Current learned string indexing approaches [91, 100, 107] optimize for variable-length storage and full/prefix matches, but they fail to support arbitrary LIKE wildcard patterns (e.g., `%keyword%`). This limitation arises from the complex challenge of modeling substring distributions under variable wildcard configurations. Our solution bridges this gap by adapting neural sequence-to-sequence models from NLP [8, 51, 92, 99]. Rather than relying on recursive tree structures, our approach resembles neural machine translation, transforming complex LIKE wildcard templates into candidates that are easy to verify. This also fundamentally differs from generative retrieval in NLP [42], which generates hypothetical documents for high-quality dense retrieval; in contrast, our work performs constrained database-specific generation to accelerate matching of complex LIKE patterns.

Efficient LIKE Processing & Approximate String Matching: The LIKE predicate is ubiquitous in databases [7, 52, 59] but hard to optimize, especially for wildcard-surrounded patterns (e.g., "%keyword%") [29, 96]. While B^+-trees support prefix/suffix matches [23, 38], n-gram-based indexes (e.g., trigrams) are commonly used for general patterns via inverted [52, 77, 114] or tree structures [29]. Approximate substring matching, often based on edit distance [10, 16, 43, 62, 64, 106], addresses vague queries in interactive systems [9, 18, 49, 61]. Unlike prior work focusing on fixed-threshold, wildcard-free patterns [62], we accelerate LIKE execution by adopting Approximate Query Processing principles [15, 76], trading completeness for speed. Our method returns a high-quality subset of guaranteed-correct matches, avoiding costly full scans on string columns when prefix-index optimization fails, especially given LIKE's prevalence in benchmarks like TPC-H (Q2,9,13,14,20) [97].

8 Conclusion

This paper introduces SMILE, a novel framework that leverages a small language model to accelerate complex LIKE queries with complex wildcards. Our evaluation shows that SMILE outperforms traditional indexing structures by up to three orders of magnitude in execution time and remains robust to shifts in both data and query distributions, highlighting its potential for modern DBMS. Our work suggests several promising directions for future research. First, the core idea: casting complex query execution as neural translation—can be extended beyond wildcard matching to other costly operations, such as regular expression evaluation and similarity joins. Second, while our current transformer-based SLM is effective against LLMs, more lightweight or database-native architectures may reduce inference overhead, particularly for high-entropy workloads where existing decoding struggles. In the future, we aim to develop more effective learned structures, enabling controllable trade-offs between performance and accuracy across diverse situations.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (NSFC) under Grant Nos. 62232005 and 62202126.

References

[1] 2023. Information technology – Database languages – SQL – Part 2: Foundation (SQL/Foundation). https://webstore.iec.ch/en/publication/86046

[2] 2025. morfessor. https://morfessor.readthedocs.io/en/latest/ Accessed: 2025-10-31.

[3] 2025. Pyphen. https://pyphen.org/ Accessed: 2025-10-31.

[4] 2025. SMILE: Source Code and Appendix. https://github.com/LiYingZe/SMILE. Accessed: 2025-12-29.

[5] Gene M. Amdahl. 1967. Validity of the Single-Processor Approach to Achieving Large Scale Computing Capabilities. In Proceedings of the AFIPS Spring Joint Computer Conference, Vol. 30. AFIPS Press, 483–485.

[6] AutoDL. 2024. AutoDL GPU Marketplace. https://www.autodl.com/market/list Accessed: 2024-06-15.

[7] Mehmet Aytimur, Silvan Reiner, Leonard Wörteler, Theodoros Chondrogiannis, and Michael Grossniklaus. 2024. LPLM: A Neural Language Model for Cardinality Estimation of LIKE-Queries. Proc. ACM Manag. Data 2, 1, Article 54 (March 2024), 25 pages. doi:10.1145/3639309

[8] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2016. Neural Machine Translation by Jointly Learning to Align and Translate. (2016). arXiv:1409.0473 [cs.CL] https://arxiv.org/abs/1409.0473

[9] Holger Bast, Debapriyo Majumdar, Ralf Schenkel, Martin Theobald, and Gerhard Weikum. 2006. IO-Top-k: index-access optimized top-k query processing. In Proceedings of the 32nd International Conference on Very Large Data Bases (Seoul, Korea) (VLDB '06). VLDB Endowment, 475–486.

[10] Alexander Behm, Shengyue Ji, Chen Li, and Jiaheng Lu. 2009. Space-Constrained Gram-Based Indexing for Efficient Approximate String Search. In Proceedings of the 2009 IEEE International Conference on Data Engineering (ICDE '09). IEEE Computer Society, USA, 604–615. doi:10.1109/ICDE.2009.32

[11] Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, Shyamal Buch, Dallas Card, Rodrigo Castellon, Niladri Chatterji, Annie Chen, Kathleen Creel, Jared Quincy Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren Gillespie, Karan Goel, Noah Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, Omar Khattab, Pang Wei Koh, Mark Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak Narayanan, Ben Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, Julian Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Rob Reich, Hongyu Ren, Frieda Rong, Yusuf Roohani, Camilo Ruiz, Jack Ryan, Christopher Ré, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishnan Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. 2022. On the Opportunities and Risks of Foundation Models. (2022). arXiv:2108.07258 [cs.LG] https://arxiv.org/abs/2108.07258

[12] Felix Brauer, Ralf Rieger, Adrian Mocan, Craig A. Knoblock, and Pedro Szekely. 2011. Enabling Information Extraction by Inference of Regular Expressions from Sample Entities. In CIKM. 1285–1290.

[13] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. arXiv preprint arXiv:2005.14165 (2020).

[14] Chen Chai, Lei Cao, Guoliang Li, Samuel Madden, and Nan Tang. 2020. Human-in-the-loop Outlier Detection. In SIGMOD. 19–33.

[15] Kaushik Chakrabarti, Minos Garofalakis, Rajeev Rastogi, and Kyuseok Shim. 2000. Approximate Query Processing Using Wavelets. In Proceedings of the 26th International Conference on Very Large Data Bases (VLDB). Morgan Kaufmann, 111–122.

[16] S. Chaudhuri, K. Ganjam, V. Ganti, and R. Motwani. 2003. Robust and Efficient Fuzzy Match for Online Data Cleaning. In Proceedings of the 2003 ACM SIGMOD International Conference on Management of Data (SIGMOD). 313–324.

[17] Surajit Chaudhuri and Luis Gravano. 1999. Evaluating Top-k Selection Queries. In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 397–410.

[18] S. Chaudhuri and R. Kaushik. 2009. Extending Autocompletion to Tolerate Errors. In Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data (SIGMOD '09). ACM. doi:10.1145/1559845.1559868

[19] Hui Chen, Xiaoyan Liu, Dingxiao Yin, and Jie Tang. 2017. A survey on dialogue systems: Recent advances and new frontiers. Acm Sigkdd Explorations Newsletter 19, 2 (2017), 25–35.

[20] Qian Chen, Arka Banerjee, Çağatay Demiralp, Samuel Madden, and Nesime Tatbul. 2023. Data Extraction via Semantic Regular Expression Synthesis. Proc. ACM Program. Lang. 7, OOPSLA2 (2023), 1848–1877.

[21] Cherry Servers. 2024. AMD EPYC 9654 Dedicated Server Pricing. https://www.cherryservers.com/pricing/dedicated-servers/amd-epyc-9654?billing=3&region=lt-siauliai

[22] Michele Dallachiesa, Ahmed Ebaid, Ahmed Eldawy, Ahmed Elmagarmid, Mourad Ouzzani, Dimitris Papadias, and Nan Tang. 2013. NADEEF: A Commodity Data Cleaning System. In SIGMOD. 541–552.

[23] Rene De La Briandais. 1959. File searching using variable length keys. In Papers Presented at the March 3-5, 1959, Western Joint Computer Conference (San Francisco, California) (IRE-AIEE-ACM '59 (Western)). Association for Computing Machinery, New York, NY, USA, 295–298. doi:10.1145/1457838.1457895

[24] DeepSeek-AI. 2024. DeepSeek-V2: A Strong, Economical, and Efficient Mixture-of-Experts Language Model. arXiv:2405.04434 [cs.CL]

[25] DeepSeek-AI. 2025. DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning. arXiv:2501.12948 [cs.CL] https://arxiv.org/abs/2501.12948

[26] Dong Deng, Guoliang Li, and Jianhua Feng. 2012. An Efficient Trie-based Method for Approximate Entity Extraction with Edit-Distance Constraints. In Proceedings of the 2012 IEEE 28th International Conference on Data Engineering (ICDE '12). IEEE Computer Society, USA, 762–773. doi:10.1109/ICDE.2012.29

[27] Jacob Devlin, Michael-W Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018).

[28] Jialin Ding, Ryan Marcus, Andreas Kipf, Vikram Nathan, Aniruddha Nrusimha, Kapil Vaidya, Alexander van Renen, and Tim Kraska. 2022. SageDB: An Instance-Optimized Data Analytics System. Proc. VLDB Endow. 15, 13 (Sept. 2022), 4062–4078. doi:10.14778/3565838.3565857

[29] PostgreSQL Documentation. [n. d.]. F.33. pg_trgm — support for similarity of text using trigram matching. https://www.postgresql.org/docs/current/pgtrgm.html. Accessed: 2025-03-07.

[30] Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A Survey on In-context Learning. In Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen (Eds.). Association for Computational Linguistics, Miami, Florida, USA, 1107–1128. doi:10.18653/v1/2024.emnlp-main.64

[31] Eduard Dragut, Fang Fang, Prasad Sistla, Clement Yu, and Weiyi Meng. 2009. Stop word and related problems in web interface integration. Proc. VLDB Endow. 2, 1 (Aug. 2009), 349–360. doi:10.14778/1687627.1687667

[32] Lynne G Duncan. 2018. Language and Reading: the Role of Morpheme and Phoneme Awareness. Current Developmental Disorders Reports 5, 4 (2018), 226–234. doi:10.1007/s40474-018-0153-2

[33] DBA Stack Exchange. [n. d.]. LIKE Query Optimization. https://dba.stackexchange.com/questions/203748/like-query-optimization. Accessed: 2025-03-07.

[34] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining small language models and large language models for zero-shot NL2SQL. Proceedings of the VLDB Endowment 17, 11 (2024), 2750–2763.

[35] Ju Fan and Guoliang Li. 2018. Human-in-the-loop Rule Learning for Data Integration. IEEE Data Eng. Bull. 41, 2 (2018), 104–115.

[36] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, and Wenyuan Yu. 2011. Interaction between record matching and data repairing. (2011), 469–480. doi:10.1145/1989323.1989373

[37] Zhangyin Feng, Daya Guo, Duyu Tang, Nan Duan, Xiaocheng Feng, Ming Gong, Linjun Shou, Bing Qin, Ting Liu, Daxin Jiang, and Ming Zhou. 2020. CodeBERT: A Pre-Trained Model for Programming and Natural Languages. (2020). arXiv:2002.08155 [cs.CL] https://arxiv.org/abs/2002.08155

[38] Paolo Ferragina and Roberto Grossi. 1999. The string B-tree: a new data structure for string search in external memory and its applications. J. ACM 46, 2 (March 1999), 236–280. doi:10.1145/301970.301973

[39] Robert M. French. 1999. Catastrophic Forgetting in Connectionist Networks. Trends in Cognitive Sciences 3, 4 (1999), 128–135.

[40] Victoria Fromkin, Robert Rodman, and Nina Hyams. [n. d.]. An Introduction to Language. Wadsworth Cengage Learning, Boston.

[41] Alex Galakatos, Michael Markovitch, Carsten Binnig, Rodrigo Fonseca, and Tim Kraska. 2019. Fiting-tree: A data-aware index structure. In Proceedings of the 2019 International Conference on Management of Data. 1189–1206.

[42] Luyu Gao, Xueguang Ma, Jimmy Lin, and Jamie Callan. 2023. 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), Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 1762–1777. doi:10.18653/v1/2023.acl-long.99

[43] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, N. Koudas, S. Muthukrishnan, and D. Srivastava. 2001. Approximate String Joins in a Database (Almost) for Free. In Proceedings of the 27th International Conference on Very Large Data Bases

(VLDB). 491–500.
[44] Max Grusky, Mor Naaman, and Yoav Artzi. 2018. Newsroom: A Dataset of 1.3 Million Summaries with Diverse Extractive Strategies. 708–719 pages. doi:10.18653/v1/N18-1065
[45] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality estimation in DBMS: a comprehensive benchmark evaluation. Proc. VLDB Endow. 15, 4 (dec 2021), 752–765. doi:10.14778/3503585.3503586
[46] Yunzhong He, Yuxin Tian, Mengjiao Wang, Feier Chen, Licheng Yu, Maolong Tang, Congcong Chen, Ning Zhang, Bin Kuang, and Arul Prakash. 2023. Que2Engage: Embedding-based Retrieval for Relevant and Engaging Products at Facebook Marketplace. In Companion Proceedings of the ACM Web Conference 2023 (Austin, TX, USA) (WWW '23 Companion). Association for Computing Machinery, New York, NY, USA, 386–390. doi:10.1145/3543873.3584633
[47] IBM Corporation 2023. IBM Db2 13 for z/OS SQL Reference. IBM Corporation. Sec. "LIKE predicate", https://www.ibm.com/docs/en/SSEPEK_13.0.0/pdf/db2z_13_sqlrefbook.pdf.
[48] IMDB. [n. d.]. Latest IMDB-Name(2025). https://datasets.imdbws.com/. Accessed: 2025-10-21.
[49] S. Ji, G. Li, C. Li, and J. Feng. 2009. Efficient Interactive Fuzzy Keyword Search. In Proceedings of the 18th International Conference on World Wide Web (WWW '09). ACM. doi:10.1145/1526709.1526718
[50] Liang Jin and Chen Li. 2005. Selectivity estimation for fuzzy string predicates in large data sets. In Proceedings of the 31st International Conference on Very Large Data Bases (Trondheim, Norway) (VLDB '05). VLDB Endowment, 397–408.
[51] Melvin Johnson, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2017. Google's Multilingual Neural Machine Translation System: Enabling Zero-Shot Translation. Transactions of the Association for Computational Linguistics 5 (2017), 339–351. doi:10.1162/tacl_a_00065
[52] Younghoon Kim, Hyoungmin Park, Kyuseok Shim, and Kyoung-Gu Woo. 2013. Efficient processing of substring match queries with inverted variable-length gram indexes. Information Sciences 244 (2013), 119–141. doi:10.1016/j.ins.2013.04.037
[53] Younghoon Kim and Kyuseok Shim. 2013. Efficient top-k algorithms for approximate substring matching. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (New York, New York, USA) (SIGMOD '13). Association for Computing Machinery, New York, NY, USA, 385–396. doi:10.1145/2463676.2465324
[54] Andreas Kipf, Thomas Kipf, Bernhard Radke, Viktor Leis, Peter Boncz, and Alfons Kemper. 2018. Learned Cardinalities: Estimating Correlated Joins with Deep Learning. (2018). arXiv:1809.00677 [cs.DB] https://arxiv.org/abs/1809.00677
[55] James Kirkpatrick, Razvan Pascanu, Neil Rabinowitz, Joel Veness, Guillaume Desjardins, Andrei A. Rusu, Kieran Milan, John Quan, Tiago Ramalho, Agnieszka Grabska-Barwinska, Demis Hassabis, Claudia Clopath, Dharshan Kumaran, and Raia Hadsell. 2017. Overcoming Catastrophic Forgetting in Neural Networks. Proceedings of the National Academy of Sciences 114, 13 (2017), 3521–3526.
[56] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In Proceedings of the 2018 international conference on management of data. 489–504.
[57] Ani Kristo, Kapil Vaidya, Ugur Çetintemel, Sanchit Misra, and Tim Kraska. 2020. The Case for a Learned Sorting Algorithm. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (Portland, OR, USA) (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 1001–1016. doi:10.1145/3318464.3389752
[58] Toshitaka Kuwa, Shigehiko Schamoni, and Stefan Riezler. 2020. Embedding Meta-Textual Information for Improved Learning to Rank. In Proceedings of the 28th International Conference on Computational Linguistics, Donia Scott, Nuria Bel, and Chengqing Zong (Eds.). International Committee on Computational Linguistics, Barcelona, Spain (Online), 5558–5568. doi:10.18653/v1/2020.coling-main.487
[59] Suyong Kwon, Kyuseok Shim, and Woohwan Jung. 2025. Cardinality Estimation of LIKE Predicate Queries using Deep Learning. Proc. ACM Manag. Data 3, 1, Article 20 (Feb. 2025), 26 pages. doi:10.1145/3709670
[60] Hongrae Lee, Raymond T. Ng, and Kyuseok Shim. 2009. Approximate substring selectivity estimation. In Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology (Saint Petersburg, Russia) (EDBT '09). Association for Computing Machinery, New York, NY, USA, 827–838. doi:10.1145/1516360.1516455
[61] Chengkai Li, Kevin Chen-Chuan Chang, Ihab F. Ilyas, and Sumin Song. 2005. RankSQL: query algebra and optimization for relational top-k queries. In Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data (Baltimore, Maryland) (SIGMOD '05). Association for Computing Machinery, New York, NY, USA, 131–142. doi:10.1145/1066157.1066173
[62] Chen Li, Bin Wang, and Xiaochun Yang. 2007. VGRAM: improving performance of approximate queries on string collections using variable-length grams. In Proceedings of the 33rd International Conference on Very Large Data

Bases (Vienna, Austria) (VLDB '07). VLDB Endowment, 303–314.
[63] Guoliang Li. 2017. Human-in-the-loop Data Integration. Proc. VLDB Endow. 10, 12 (2017), 2006–2017.
[64] Guoliang Li, Dong Deng, and Jianhua Feng. 2011. Faerie: efficient filtering algorithms for approximate dictionary-based entity extraction. In Proceedings of the 2011 ACM SIGMOD International Conference on Management of Data (Athens, Greece) (SIGMOD '11). Association for Computing Machinery, New York, NY, USA, 529–540. doi:10.1145/1989323.1989379
[65] Guoliang Li, Shengyue Ji, Chen Li, and Jianhua Feng. 2011. Efficient fuzzy full-text type-ahead search. The VLDB Journal 20, 4 (August 2011), 617–640. doi:10.1007/s00778-011-0218-x
[66] Pengfei Li, Wenqing Wei, Rong Zhu, Bolin Ding, Jingren Zhou, and Hua Lu. 2023. ALECE: An Attention-based Learned Cardinality Estimator for SPJ Queries on Dynamic Workloads. Proc. VLDB Endow. 17, 2 (oct 2023), 197–210. doi:10.14778/3626292.3626302
[67] Yingze Li, Hongzhi Wang, and Xianglong Liu. 2024. One Seed, Two Birds: A Unified Learned Structure for Exact and Approximate Counting. Proc. ACM Manag. Data 2, 1, Article 15 (mar 2024), 26 pages. doi:10.1145/3639270
[68] Zeyu Li, Hongzhi Wang, Wei Shao, Jianzhong Li, and Hong Gao. 2016. Repairing Data through Regular Expressions. Proc. VLDB Endow. 9, 5 (2016), 432–443. doi:10.14778/2876473.2876478
[69] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. arXiv preprint arXiv:1907.11692 (2019).
[70] Yurong Liu, Eduardo Pena, Aecio Santos, Eden Wu, and Juliana Freire. 2025. Magneto: Combining Small and Large Language Models for Schema Matching. (2025). arXiv:2412.08194 [cs.DB] https://arxiv.org/abs/2412.08194
[71] Microsoft 2023. LIKE (Transact-SQL). Microsoft. https://learn.microsoft.com/en-us/sql/t-sql/language-elements/like-transact-sql?view=sql-server-ver17.
[72] Sewon Min, Mike Lewis, Luke Zettlemoyer, and Hannaneh Hajishirzi. 2022. MetaICL: Learning to Learn In Context. In Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 2791–2809. doi:10.18653/v1/2022.naacl-main.201
[73] Colin Morris. 2017. Reddit Usernames. https://www.kaggle.com/datasets/colinmorris/reddit-usernames. Accessed: 2025-03-18.
[74] John X. Morris, Chawin Sitawarin, Chuan Guo, Narine Kokhlikyan, G. Edward Suh, Alexander M. Rush, et al. 2025. How much do language models memorize? arXiv preprint arXiv:2505.24832 (2025). https://arxiv.org/abs/2505.24832
[75] Brent Ozar. [n. d.]. Sargability: Why %string% Is Slow. https://www.brentozar.com/archive/2010/06/sargable-why-string-is-slow/. Accessed: 2025-03-07.
[76] Jialin Peng, Donghui Zhang, and Feifei Li. 2018. AQP++: Connecting Approximate Query Processing With Aggregate Precomputation for Interactive Analytics. In Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data. ACM, 1093–1107.
[77] PostgreSQL Core Team. 2025. GIN Indexes. https://www.postgresql.org/docs/current/gin.html. Accessed: 2025-03-18.
[78] Abdullah Qahtan, Nan Tang, Mourad Ouzzani, and Michael Stonebraker. 2020. Pattern Functional Dependencies for Data Cleaning. Proc. VLDB Endow. 13, 5 (2020), 684–697.
[79] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. (2018).
[80] Alec Radford, Jeffrey Wu, Rewon Child, David Chen, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. OpenAI blog 1, 8 (2019).
[81] Vijayshankar Raman and Joseph M. Hellerstein. 2001. Potter's Wheel: An Interactive Data Cleaning System. In VLDB. 381–390.
[82] David Rolnick, Arun Ahuja, Jonathan Schwarz, Timothy P. Lillicrap, and Gregory Wayne. 2019. Experience Replay for Continual Learning. NeurIPS 32 (2019).
[83] Olli Räsänen, Reima Laaksonen, and Mikko Kurimo. 2018. Pre-linguistic segmentation of speech into syllable-like units. Cognition 171 (2018), 130–150. doi:10.1016/j.cognition.2017.11.003
[84] Ibrahim Sabek, Kapil Vaidya, Dominik Horn, Andreas Kipf, Michael Mitzenmacher, and Tim Kraska. 2022. Can Learned Models Replace Hash Functions? Proc. VLDB Endow. 16, 3 (Nov. 2022), 532–545. doi:10.14778/3570690.3570702
[85] SAP SE 2023. SAP IQ 16.0 SP04 Reference: Building Blocks. SAP SE. PDF p. 80, https://infocenter.sybase.com/help/topic/com.sybase.infocenter.dc38151.1604/doc/pdf/iqrefbb.pdf.
[86] Gaurav Saxena, Mohammad Rahman, Naresh Chainani, Chunbin Lin, George Caragea, Fahim Chowdhury, Ryan Marcus, Tim Kraska, Ippokratis Pandis, and Balakrishnan (Murali) Narayanaswamy. 2023. Auto-WLM: Machine Learning Enhanced Workload Management in Amazon Redshift. In Companion of the 2023 International Conference on Management of Data (Seattle, WA, USA) (SIGMOD '23). Association for Computing Machinery, New York, NY, USA, 225–237. doi:10.1145/3555041.3589677

[87] Anil Shanbhag, Samuel Madden, and Xiangyao Yu. 2020. A Study of the Fundamental Performance Characteristics of GPUs and CPUs for Database Analytics. In Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data (Portland, OR, USA) (SIGMOD '20). Association for Computing Machinery, New York, NY, USA, 1617–1632. doi:10.1145/3318464.3380595

[88] Saurabh Shetiya, Saravanan Thirumuruganathan, Nick Koudas, and Divesh Srivastava. 2020. Astrid: Accurate Selectivity Estimation for String Predicates Using Deep Learning. Proc. VLDB Endow. 14, 4 (2020), 684–697.

[89] Koustuv Sinha, Robin Jia, Dieuwke Hupkes, Joelle Pineau, Adina Williams, and Douwe Kiela. 2021. Masked Language Modeling and the Distributional Hypothesis: Order Word Matters Pre-training for Little. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 2888–2913. doi:10.18653/v1/2021.emnlp-main.230

[90] Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM Test-Time Compute Optimally Can Be More Effective Than Scaling Model Parameters. arXiv:2408.03314 [cs.CL]

[91] Benjamin Spector, Andreas Kipf, Kapil Vaidya, Chi Wang, Umar Farooq Minhas, and Tim Kraska. 2021. Bounding the Last Mile: Efficient Learned String Indexing. In AIDB.

[92] Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. 2014. Sequence to sequence learning with neural networks. In Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2 (Montreal, Canada) (NIPS'14). MIT Press, Cambridge, MA, USA, 3104–3112.

[93] DeepSeek Team. 2024. DeepSeek-V3 Technical Report. https://arxiv.org/html/2412.19437v1 arXiv preprint arXiv:2412.19437v1.

[94] Qwen Team. 2025. Qwen2.5 Technical Report. Technical Report 2412.15115. arXiv. https://arxiv.org/pdf/2412.15115

[95] Together Computer. 2023. RedPajama-Data-1T-Sample: A 1.2 B-token sample of the RedPajama pre-training corpus. https://huggingface.co/datasets/togethercomputer/RedPajama-Data-1T-Sample Hugging Face dataset snapshot, accessed 2025-10-24.

[96] Ask TOM. [n. d.]. Using the LIKE predicate in a WHERE clause. https://asktom.oracle.com/ords/asktom.search?tag=using-the-like-predicate-in-a-where-clause. Accessed: 2025-03-07.

[97] TPC. [n. d.]. TPC-H Homepage. https://www.tpc.org/tpch/. Accessed: 2025-03-18.

[98] Kapil Vaidya, Subarna Chatterjee, Eric Knorr, Michael Mitzenmacher, Stratos Idreos, and Tim Kraska. 2022. SNARF: a learning-enhanced range filter. Proc. VLDB Endow. 15, 8 (April 2022), 1632–1644. doi:10.14778/3529337.3529347

[99] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In Proceedings of the 31st International Conference on Neural Information Processing Systems (Long Beach, California, USA) (NIPS'17). Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[100] Youyun Wang, Chuzhe Tang, Zhaoguo Wang, and Haibo Chen. 2020. SIndex: a scalable learned index for string keys. In Proceedings of the 11th ACM SIGOPS Asia-Pacific Workshop on Systems (Tsukuba, Japan) (APSys '20). Association for Computing Machinery, New York, NY, USA, 17–24. doi:10.1145/3409963.3410496

[101] WIKI. [n. d.]. Latest WIKI-dump(2025). https://dumps.wikimedia.org/enwiki/. Accessed: 2025-10-21.

[102] Jiacheng Wu, Yong Zhang, Shimin Chen, Jin Wang, Yu Chen, and Chunxiao Xing. 2021. Updatable Learned Index with Precise Positions. Proc. VLDB Endow. 14, 8 (apr 2021), 1276–1288. doi:10.14778/3457390.3457393

[103] Shiguang Wu, Yaqing Wang, and Quanming Yao. 2025. Why In-Context Learning Models are Good Few-Shot Learners?. In The Thirteenth International Conference on Learning Representations. https://openreview.net/forum?id=iLUcsecZJp

[104] Ziniu Wu, Ryan Marcus, Zhengchun Liu, Parimarjan Negi, Vikram Nathan, Pascal Pfeil, Gaurav Saxena, Mohammad Rahman, Balakrishnan Narayanaswamy, and Tim Kraska. 2024. Stage: Query Execution Time Prediction in Amazon Redshift. In Companion of the 2024 International Conference on Management of Data (Santiago AA, Chile) (SIGMOD/PODS '24). Association for Computing Machinery, New York, NY, USA, 280–294. doi:10.1145/3626246.3653391

[105] Chuan Xiao, Wei Wang, Xuemin Lin, Jeffrey Xu Yu, and Guoren Wang. 2011. Efficient similarity joins for near-duplicate detection. ACM Trans. Database Syst. 36, 3, Article 15 (Aug. 2011), 41 pages. doi:10.1145/2000824.2000825

[106] Xiaochun Yang, Bin Wang, and Chen Li. 2008. Cost-based variable-length-gram selection for string collections to support approximate queries efficiently. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (Vancouver, Canada) (SIGMOD '08). Association for Computing Machinery, New York, NY, USA, 353–364. doi:10.1145/1376616.1376655

[107] Yifan Yang and Shimin Chen. 2024. LITS: An Optimized Learned Index for Strings. Proc. VLDB Endow. 17, 11 (July 2024), 3415–3427. doi:10.14778/3681954.3682010

[108] Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. 2020. NeuroCard: One Cardinality Estimator for All Tables. Proc. VLDB Endow. 14, 1 (sep 2020), 61–73. doi:10.14778/3421424.3421432

[109] Zongheng Yang, Eric Liang, Amog Kamsetty, Chenggang Wu, Yan Duan, Xi Chen, Pieter Abbeel, Joseph M. Hellerstein, Sanjay Krishnan, and Ion Stoica. 2019. Deep Unsupervised Cardinality Estimation. Proc. VLDB Endow. 13, 3 (nov 2019), 279–292. doi:10.14778/3368289.3368294

[110] Shaoxiong Yu, Li Han, Marta Indulska, and Hongzhi Liu. 2023. Human-in-the-loop Regular Expression Extraction for Single Column Format Inconsistency. In WWW. 3859–3867.

[111] Yicheng Zhang, Yuhao Wang, Ziyu Zhang, Yiming Zhang, Yizhe Zhang, Haoyang Zhang, Shengyu Zhang, Yifan Zhang, Yuchen Zhang, Yuxuan Zhang, et al. 2023. Math Problem Solving with Large Language Models: A Survey. arXiv preprint arXiv:2305.00772 (2023).

[112] Fan Zhou and Chen Cao. 2021. Overcoming Catastrophic Forgetting in Graph Neural Networks with Experience Replay. In AAAI, Vol. 35. 4714–4722.

[113] Yuqi Zhu, Jia Li, Ge Li, YunFei Zhao, Jia Li, Zhi Jin, and Hong Mei. 2024. Hot or Cold? Adaptive Temperature Sampling for Code Generation with Large Language Models. Proceedings of the AAAI Conference on Artificial Intelligence 38, 1 (Mar 2024), 437–445. doi:10.1609/aaai.v38i1.27798

[114] Justin Zobel and Alistair Moffat. 2006. Inverted files for text search engines. ACM Comput. Surv. 38, 2 (July 2006), 6–es. doi:10.1145/1132956.1132959

Received July 2025; revised October 2025; accepted November 2025