# RIVET: Regex-to-Indexable Keys via Neural Translation for Interactive LIMIT-$k$ Retrieval

**Dong Wang**[1,†] **Yingze Li**[1,†] **Zihang Yu**[1] **Ziqing Zeng**[1]
**Zongdai Jiang**[1] **Yuhao Zhang**[1] **Kaixin Zhang**[1]
**Shuangshuang Cui**[1] **Hongzhi Wang**[1,*]
[1]Harbin Institute of Technology
[†]Equal contribution. [*]Corresponding author: wangzh@hit.edu.cn.

## Abstract

Interactive regex search over large text columns needs exact matches under low latency, but indexes help only when a pattern exposes selective fragments. For LIMIT-$k$ queries, the immediate bottleneck is choosing a few indexable keys that reach match-rich candidate buckets before regex verification. RIVET casts this step as constrained regex-to-key generation: a column-specific translator emits short keys, an existing index retrieves candidates, and the native regex engine verifies every output record. Training from repeated regex-key hits moves probability mass toward effective keys, so finite online sampling spends fewer regex checks before collecting the requested matches. Across four large datasets, RIVET reaches 35.8 ms median latency, improves over the strongest baseline REI by 3.4–36×, and reaches up to 1773× speedup over sequential scan. The same mechanism improves TPC-H query plans, preview retrieval under latency budgets, and 80 human-authored OOD regex queries, showing that regex-to-key generation translates into end-to-end system gains.

## 1 Introduction

Regular expressions (regexes) are a standard interface for exact text search in large information systems Majumder et al. [2008], Zhang et al. [2023]. A user may search logs, names, titles, addresses, or error messages with a pattern, and the returned records must satisfy that pattern exactly. Many such queries are interactive: the user asks for a small preview, such as the first 10 or 64 matches, and then revises the pattern or continues the analysis. This puts the system under a simple pressure: obtain enough verified matches quickly. When a regex exposes a clear prefix, literal, or rare substring, existing indexes can narrow the search space before verification. When the pattern contains wildcards, character classes, alternation, or repetition, the useful index entry is harder to read directly from the surface form of the regex; the system may spend most of its time verifying low-yield candidates even under a LIMIT-$k$ query Yang et al. [2013], Riveros et al. [2023], Carey and Kossmann [1997], Chaudhuri and Gravano [1999], Chang and Hwang [2002]. The subproblem we study is therefore *which indexable keys a regex query should use first*.

**The Gap.** Three lines of work set up this question. Regex engines and automata-based execution provide the exact per-record verifier: a candidate record is returned only after the engine confirms that the pattern appears in the string Yang et al. [2016], Riveros et al. [2023], Cox [2010], Wang et al. [2019]. Index-based regex acceleration reduces the number of records sent to this verifier by extracting literals, prefixes, trigrams, $n$-grams, or signatures from the pattern and using them as retrieval keys Cho and Rajagopalan [2002], Kim et al. [2005], Yang et al. [2008], PostgreSQL Global Development Group [2024]. Learned database components show that data distributions can be modeled and used inside query processing, especially for indexes and selectivity

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

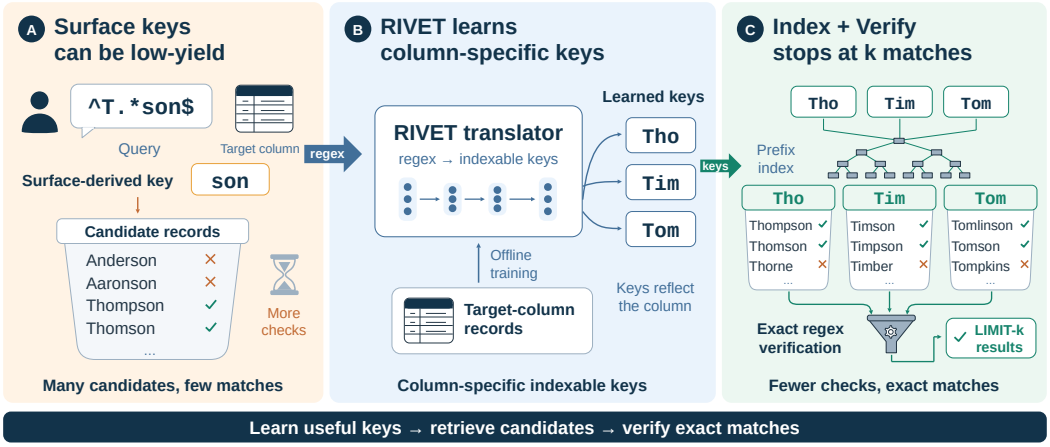



Figure 1: RIVET casts interactive LIMIT-$k$ regex retrieval as constrained regex-to-key generation. **(A)** Keys read off the regex surface are column-agnostic and often low-yield. **(B)** RIVET's column-specific translator emits indexable keys that need not appear in the regex. **(C)** A Key+Verify executor probes the index, verifies with the regex engine, and stops once $k$ matches are found.

estimation Kraska et al. [2018], Ding et al. [2020], Hilprecht et al. [2020], Yang et al. [2020], Shetiya et al. [2020]. *For interactive regex retrieval, the missing step is to generate a small set of executable keys for a given regex and a given text column.* Counting how many rows may match is useful for planning, but a LIMIT-$k$ executor also needs concrete keys that point to high-yield candidate buckets.

**Our core insight.** *Interactive regex retrieval can be cast as constrained regex-to-key generation.* Values in one text column often share formats, templates, and repeated fragments. The same regex can therefore have different useful keys on different columns. A prefix that is selective for names may be common in logs, and a literal that appears in the regex may be less useful than another short key implied by the column distribution. RIVET learns this relation offline from the target column and, at query time, generates a bounded set of indexable keys. The database index uses these keys to retrieve candidates, and the native regex engine verifies the candidates before output. The mechanism is direct: when generated keys point to buckets with more true matches, the system spends fewer regex checks before collecting the requested matches. Our analysis formalizes this through effective keys: increasing the model probability mass on keys that cover true matches increases the chance that a fixed number of samples reaches a useful bucket.

**Contributions.** We make the following contributions:

❶ **Regex-to-key formulation.** We formulate interactive LIMIT-$k$ regex retrieval as constrained generation from regex patterns to a bounded set of indexable retrieval keys.

❷ **RIVET framework.** We build a column-specific translator that learns from weak regex-key supervision, generates diverse keys under an online budget, and connects those keys to a Key+Verify execution pipeline.

❸ **Coverage-cost analysis.** We analyze the generate-and-verify process through effective keys, showing how probability mass on keys that cover true matches controls the chance of reaching useful candidate buckets under finite sampling.

❹ **System validation.** Across large text columns, RIVET reaches 26.3–71.3 ms latency for single-column LIMIT-64 retrieval, gives 3.38×–36.15× speedups over REI, and improves end-to-end TPC-H plans and preview retrieval under interactive budgets.

## 2 Background and Motivation

Regex processing starts from an exact verifier. Given a pattern and a candidate record, a regex engine checks whether some substring of the record is accepted by the pattern. Automata-based engines and production libraries make this per-record check efficient and reliable Yang et al. [2016], Riveros et al. [2023], Cox [2010], Wang et al. [2019]. In a large text column, however, the total cost is

governed by how many records are sent to the verifier. Large-scale regex retrieval therefore needs a candidate-selection step before exact checking Majumder et al. [2008], Zhang et al. [2023].

### 2.1 Background

Index-based regex acceleration supplies this candidate-selection step. The usual strategy is to extract literals, prefixes, trigrams, $n$-grams, or signatures from the regex and use them to query an index Cho and Rajagopalan [2002], Kim et al. [2005], Yang et al. [2008], PostgreSQL Global Development Group [2024], Apache Software Foundation [2024]. These extracted fragments act as retrieval keys: the index fetches records associated with the keys, and the regex engine verifies the fetched records. This key-then-verify structure is the standard way to combine fast candidate access with exact regex semantics. In particular, REI Zhang et al. [2025] constructs an $n$-gram bitmask from regex literals, filters records through their $n$-gram bit-vectors, and verifies the survivors. Selective indexed grams make this filter effective; a zero query mask leaves full-table verification. REI is the fastest baseline in our main evaluation. RIVET instead learns which column-specific buckets yield matches early under a key budget.

The benefit of a retrieval key depends on the candidate bucket it retrieves. A useful key retrieves a bucket that is small enough to verify quickly and rich enough to contain matches. A frequent literal can be weak because it retrieves many records that still need regex checks. A rare prefix can be strong when it points to a compact bucket with many true matches. Complex operators such as alternation, character classes, and repetition further make useful keys hard to read from local fragments alone. Key quality is therefore tied to both the regex structure and the target column distribution Yang et al. [2013], Bast and Celikik [2013], Qiu et al. [2022].

`LIMIT-`$k$ queries change the execution target. In interactive and exploratory analysis, users often ask for a small number of matches to inspect a pattern or validate a hypothesis Carey and Kossmann [1998, 1997], Chang and Hwang [2002]. The executor needs enough verified matches quickly, and a high-hit bucket is valuable because it can satisfy the limit after few regex checks. A low-hit bucket spends the same verifier on low-value candidates. This makes retrieval-key choice a first-order factor in interactive regex latency Fagin et al. [2001], Melnik et al. [2010], Ilyas et al. [2004].

### 2.2 Motivation: the missing gap

The background above leaves a concrete question: given a regex and a target text column, how should the system obtain a small set of indexable keys that are likely to lead to verified matches? Hand-written extraction rules provide useful keys when the regex exposes clear local fragments, but the payoff of a key depends on how the pattern meets the column distribution. Text columns usually contain repeated formats and local regularities, so this regex-to-key relation can be learned from examples drawn from the same column. We therefore study constrained regex-to-key generation for `LIMIT-`$k$ regex retrieval. The generated keys are used to retrieve candidates through an index; the native regex engine then checks the candidates before any record is returned. RIVET follows this design by learning column-specific regex-to-key generation offline and using the generated keys online for fast candidate retrieval and exact verification.

**Interactive use case.** Result limits occur in large query logs Bonifati et al. [2022]. Microsoft Purview supports `matches regex` and recommends `take` to validate queries before full analysis; Kusto's equivalent `take` and `limit` operators return unordered previews Microsoft [2026, 2024]. For a Wikipedia-title query matching "List of X in Y," adapted from a public request Sharma [2020], every method returns 64 verified matches: RIVET takes 123.9 ms, REI 658.4 ms, BLARE 1,527.7 ms, and SeqScan 7,025.3 ms (App. E.1). This delay recurs with each regex revision during interactive analysis.

## 3 Preliminaries

This section fixes the retrieval interface used by RIVET. We first define the `LIMIT-`$k$ regex task, the Key+Verify execution model, and the cost metric. We then explain why key generation must depend on the target column rather than only on the regex surface form. Full formal definitions are given in App. A.

### 3.1 Problem Formulation and Interfaces

We consider standard regular expressions $Q$ over an alphabet $\Sigma$ and a dataset $D = \{t_i\}_{i=1}^{n} \subseteq \Sigma^*$. A record $t$ is a *match* if a regex engine finds any substring of $t$ that matches $Q$. Let $W(Q)$ denote the set of matching records for $Q$ (formally defined in App. A). Given $k \geq 1$, the executor should return up to $k$ *verified* matches under strict latency while minimizing expensive regex verifications. The task uses `LIMIT` without `ORDER BY`: any $k$ distinct satisfying records fill the limit, and a scan returns all matches if fewer than $k$ exist. The fixed-budget key path may return fewer records; residual scanning completes the limit when required.

RIVET uses a Key+Verify interface. At query time, a learned translator outputs an unordered key set $K_\theta(Q)$ with $|K_\theta(Q)| \leq t_{\max}$. An index returns a possibly deduplicated candidate set $\mathrm{Cand}(K_\theta(Q))$ for the generated keys, as specified by the index interface in App. A. The system then verifies candidates sequentially with a regex engine and outputs only verified matches. Recall@$k$ measures coverage under the fixed key budget $t_{\max}$, while exact verification ensures that every returned record satisfies the original regex predicate.

The index interface is defined through a key extractor $\phi : \Sigma^* \to \Sigma^\ell$. The attainable key dictionary on dataset $D$ is

$$\mathcal{K}(D) := \{\phi(t) \mid t \in D\}.$$

For a key $p \in \mathcal{K}(D)$, the index returns the bucket $\mathrm{Cand}(p) := \{t \in D \mid \phi(t) = p\}$; for a key set $P \subseteq \mathcal{K}(D)$, it returns $\mathrm{Cand}(P) := \bigcup_{p \in P} \mathrm{Cand}(p)$. In our prefix-key instantiation, each record has one prefix key, so buckets for distinct keys are disjoint. This lets candidate coverage and candidate volume decompose over the selected key set, which is the quantity the online decoder controls.

We measure efficiency by the number of regex verifications issued before termination:

$$\mathrm{cost}(Q; k) := \#\{\text{regex verifications issued before termination}\}.$$

This cost isolates the main bottleneck in `LIMIT-`$k$ regex retrieval: how many candidate records must be checked before enough matches are found.

### 3.2 Regex-to-Key Perspective

Under generate–verify execution, the central difficulty is to spend the verification budget on candidate regions that are likely to contain true matches. Conventional index-based methods act as static translators from regexes to retrieval keys: they extract literals, local fragments, prefixes, or $q$-grams from $Q$ and use those fragments to probe an index. This works when a surface fragment is both valid for the regex and selective in the target column. It weakens when the regex lacks discriminative literals, when common fragments retrieve large low-yield buckets, or when concatenation, alternation, and closure create non-local constraints that local fragments do not capture.

The usefulness of a key is therefore data-dependent. Values in a single text column often share formats, templates, and repeated fragments, so the same regex can have different high-yield keys on different columns. A fragment that is rare and useful in one column can be frequent and uninformative in another. RIVET learns this column-specific relation: instead of hard-coding a rule from $Q$ to keys, it trains a data-dependent translator that generates keys more correlated with true matches under the target column distribution. The learned component chooses short keys; the database regex engine remains responsible for predicate semantics and final exact checking.

This perspective differs from selectivity estimation. A planner-side estimate can say that a regex is likely to match many or few rows, but a `LIMIT-`$k$ executor still needs concrete access paths that can be probed immediately. RIVET therefore predicts executable keys rather than only a scalar cardinality. The generated keys are useful only insofar as their buckets contain verified matches early, which ties the learning target directly to online verification cost.

## 4 Method

### 4.1 Overview

RIVET learns to translate a regex into a small set of indexable keys that can retrieve candidates from existing indices before exact regex verification (Figure 2). Offline, the system constructs a

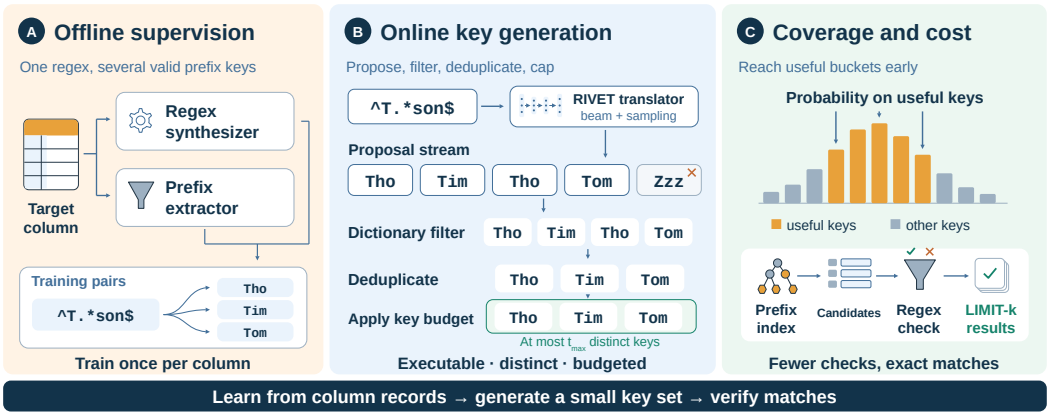



Figure 2: RIVET system overview. **Offline**, RIVET builds column-derived supervision pairing each regex with multiple prefix keys and trains a translator $f_\theta$. **Online**, $f_\theta$ produces a proposal stream that is dictionary-filtered, deduplicated, and budget-capped into a small key set $K_\theta(Q)$. The executor probes existing indices with $K_\theta(Q)$, verifies candidates with the regex engine, and stops when $k$ matches are found or the budget is exhausted.

workload from the target column $D$ and trains a translator $f_\theta$. Online, given a regex $Q$, the translator generates a key set $K_\theta(Q)$ with $|K_\theta(Q)| \leq t_{\max}$. The executor probes indices with these keys, verifies candidates with the regex engine, and reports Recall@$k$ as coverage under the fixed key budget. The objective is to return verified matches quickly while reducing the number of verifications $\mathrm{cost}(Q; k)$.

Three design choices determine whether this pipeline works. The training signal must expose multiple feasible keys for the same query, because many key sets can lead to useful candidate buckets. The translator must represent regex structure well enough to predict prefixes constrained by alternation, quantifiers, and scope boundaries. The online decoder must produce a diverse key set rather than a single best guess, since different keys cover different regions of the match set.

### 4.2 Offline Workload and Supervision

The offline stage fixes the key primitive and builds training pairs without enumerating the full feasible-key set. We instantiate indexable retrieval keys as fixed-length prefixes. Throughout this section, $\phi$ denotes the fixed-length prefix extractor with key length $\ell$, so $\phi(t)$ is the length-$\ell$ prefix key of record $t$.

Prefix keys provide a simple many-to-one access path: a single key usually maps to multiple candidate records. Under a constrained key budget, the useful operating point balances coverage and bucket selectivity. Prefix predicates also map directly to common index structures, such as B-trees or tries, through range retrieval; the regex engine then verifies retrieved candidates one by one. Exact verification preserves predicate-level soundness, while Recall@$k$ captures how much coverage the chosen prefix keys provide under the key budget.

RIVET constructs example-driven supervision by sampling an anchor record $t \sim D$, synthesizing a regex $Q$ consistent with $t$, and supervising the translator with the prefix key $p = \phi(t)$, yielding a training pair $(Q, p)$. This grounds supervision in records actually drawn from the target column, where useful keys are determined jointly by regex semantics and the prefix-bucket distribution.

The generator intentionally permits repeated regex instances. The same regex $Q$ can recur because anchor records are sampled stochastically and because different anchors can yield the same island sequence and gap pattern. When this happens, the translator observes the same $Q$ paired with multiple anchor-derived prefixes $p = \phi(t)$. These repeated hits expose multiple feasible positives for the same regex during training, providing richer supervision over valid keys and improving key-set coverage under a fixed budget. The main workloads vary two knobs: the anchor length $n \in \{2, 3\}$ and exposure budget $r \in \{0.2, 0.3\}$; all template and sampling details are in App. D.2.

### 4.3 Online Generation, Verification, and Analysis

At query time, RIVET outputs a small set of fixed-length prefix keys $K_\theta(Q) \subseteq \mathcal{K}(D)$. Each key retrieves a candidate list that must be verified by the regex engine, so the key set should cover many true matches without sending large low-yield buckets to verification.

The translator is a character-level seq2seq model from regexes to length-$\ell$ prefixes. Character-level modeling preserves regex metacharacter boundaries and lets the encoder represent non-local structure such as alternation scopes and quantifier boundaries. Given an input regex $Q$, the decoder emits a length-$\ell$ prefix $p = (p_1, \ldots, p_\ell)$ autoregressively:

$$q_{\theta,T}(p \mid Q) := \prod_{t=1}^{\ell} P_{\theta,T}(p_t \mid p_{<t}, Q).$$

During training, teacher forcing feeds the ground-truth prefix history $p_{<t}$ to the decoder and maximizes $\log q_{\theta,1}(p \mid Q)$. We use character-level tokenization rather than subword tokenization so that regex operators such as `.*`, character classes, and escaped literals remain visible at the same granularity used by the generator.

Online key-set construction combines beam proposals with temperature samples. RIVET retains proposals in the column dictionary $\mathcal{K}(D)$, deduplicates them, and keeps at most $t_{\max}$ distinct keys as $K_\theta(Q)$. Beam proposals cover high-probability prefixes, while temperature samples reach complementary buckets. Filtering keeps index probes within the column's key space, and deduplication prevents repeated proposals from consuming the key budget. Architectural details and proposal processing are in App. B.3 and App. B.1.

The executor processes the generated keys sequentially. For each prefix key, it uses an indexed prefix predicate, such as `LIKE 'p%'`, to retrieve candidates and then runs exact regex verification over those candidates. The system stops when either $k$ matches have been obtained or all candidates retrieved by the output key set have been verified. Since every output passes regex-engine verification, generated keys affect candidate sizes and hit rates, not predicate correctness.

**Completing the limit.** Preview retrieval ends with the key path. For a filled limit, residual scanning verifies the remaining records until $k$ distinct matches or column exhaustion; record identifiers prevent duplicate output (App. B.2). Section 5.4 measures the combined cost.

To analyze the stochastic component, we consider a fixed stream of independent temperature samples before budget truncation. This finite-sample view isolates the effects of learned probability mass and sampling diversity. Let $\Pi_\theta(Q)$ be the deduplicated key set from $S$ i.i.d. draws, write $h(p; Q) := |\mathrm{Cand}(p) \cap W(Q)|$ for the number of records in bucket $\mathrm{Cand}(p)$ that match $Q$, define $\mathcal{K}^+(Q) := \{p \in \mathcal{K}(D) \mid h(p; Q) > 0\}$, and let $\mu_\theta(Q) = \sum_{p \in \mathcal{K}^+(Q)} q_{\theta,T}(p \mid Q)$ be the model probability mass on effective keys. Then

$$\Pr\big(\Pi_\theta(Q) \cap \mathcal{K}^+(Q) \neq \emptyset\big) = 1 - (1 - \mu_\theta(Q))^S. \tag{1}$$

Increasing $\mu_\theta(Q)$ or the sampling budget $S$ raises the chance of reaching a hit-yielding key. In the `LIMIT-`$k$ setting, key selection reduces verification work when it directs execution toward match-rich buckets. The same view also makes the role of learned mass concrete: useful samples must put probability mass on keys that both exist in the column and cover true matches. Beyond the basic hit probability, App. C.3–C.6 develop the analysis connecting $\mu_\theta(Q)$ to system-level quantities: anchor sampling places the population-optimal supervision on $\mathcal{K}^+(Q)$ weighted by per-bucket hit counts (App. C.3); a negative-hypergeometric formula relates bucket hits and volume to expected verification cost under uniform random candidate order (App. C.4); the expected effective-key coverage is Schur-concave in $q_{\theta,T}$, describing the diversity advantage examined in App. L (App. C.5); and a Pinsker bound (App. C.6) ties the cross-entropy training risk to the online coverage gap. Start-of-record constraints and column bucket counts give a sufficient condition for selective prefix routing (App. C.7). App. C.8 reports per-column empirical validation: dictionary-hit probability calibration gives per-column mean absolute errors of $0.0001$–$0.0021$, the Schur interval contains the empirical effective-key coverage on $132/150$ queries with $|\mathcal{K}^+| > 1$ at $S = 32$, and the NHG expression agrees within $5\%$ with mean costs from random-order simulation on all 97 eligible queries.

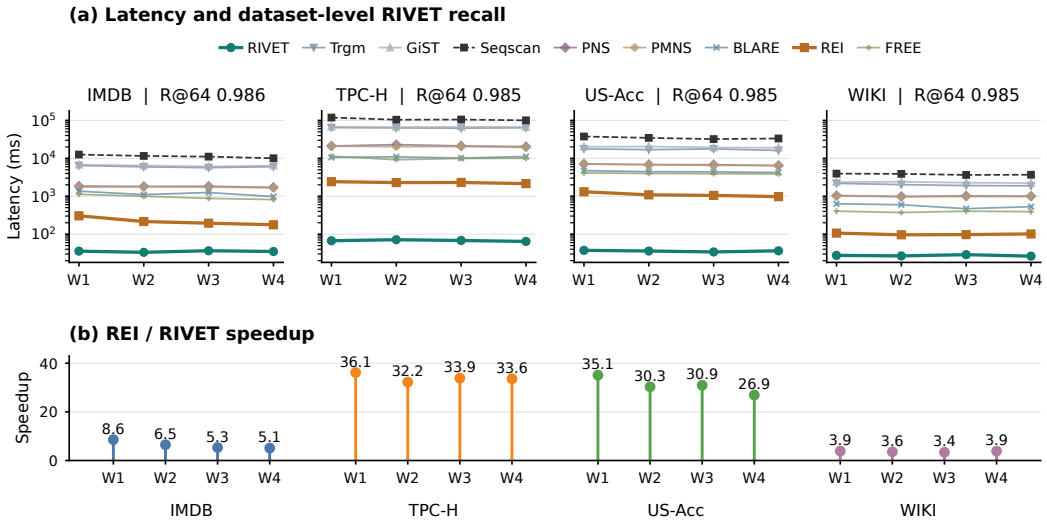



Figure 3: **Single-column `LIMIT`-64 retrieval across all datasets, workloads, and methods.** (a) Latency profiles: 4 datasets × 4 workloads × 9 methods, with log-scaled latency so millisecond-, second-, and scan-level methods remain comparable. (b) REI/RIVET speedup for each dataset–workload pair. Dataset-level RIVET Recall@64 is shown beside the latency panels.

# 5 Experiments

We evaluate retrieval quality and latency, the contribution of learned key selection, and its utility in query plans, interactive previews, and deployment.

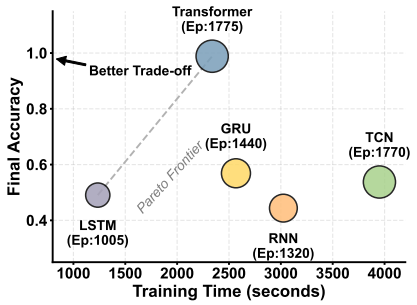
## 5.1 Experimental Setup

▼ **Setup.** We evaluate on four large text columns: IMDB names (5.0M rows), WIKI titles (2.0M), TPC-H `lineitem.comment` (24.0M), and US-Accident descriptions (7.72M). The main synthesized workloads are PostgreSQL-compatible regexes controlled by anchor length $n \in \{2, 3\}$ and exposure budget $r \in \{0.2, 0.3\}$, giving $W1 = (2, 0.2)$, $W2 = (2, 0.3)$, $W3 = (3, 0.2)$, and $W4 = (3, 0.3)$. We compare with SEQSCAN, `trgm` PostgreSQL Global Development Group [2024], GiST PostgreSQL Global Development Group [2025], FREE Cho and Rajagopalan [2002], REI Zhang et al. [2025], BLARE Zhang et al. [2023], PNS Yang et al. [2016], and PMNS-LITE Qiu et al. [2022]. Metrics are Latency@k, Recall@k for limit completion against the SEQSCAN oracle, TPC-H end-to-end latency, preview Success@$\tau$, and interactive total time / rounds. The main runs use PostgreSQL 16.3 with colocated inference; dataset statistics, baseline configurations, and metric definitions are in App. D.1–D.4. The OOD comparisons use either the same per-query output target $h(q)$ as fixed-budget RIVET or the original limit of 64 with residual scanning. CPU timings isolate translator inference; RPC measurements report payload size (App. I.4).

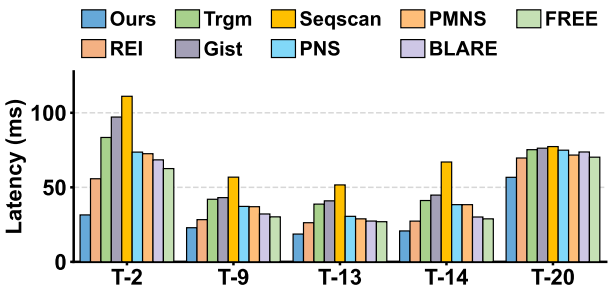
## 5.2 Mechanism and Main Retrieval Results

▼ **Exp.1. Single-column retrieval.** Figure 3 gives the main retrieval result at $k = 64$. Across datasets and workloads, RIVET runs in **26.3–71.3 ms**, with mean Recall@64 ≈**0.986**. Relative to the strongest single-column baseline REI, the average speedup is **at least 3.7× across datasets and 3.4× across workloads, reaching 33.9× and 36.2× respectively**. Relative to SEQSCAN, the speedup reaches **139×–1586×** on average by dataset. The detailed per-workload table and $k \in \{16, 32, 128\}$ results remain in App. E.2, where the same trend persists.

▼ **Exp.2. Component evidence.** Decoded keys outperform alternative key-selection strategies by orders of magnitude. With $t_{\max} = 32$, random keys from the column dictionary achieve only **0.002–0.116** Recall@64; a single greedy decode reaches **0.45–0.69**; beam search recovers **0.945–0.961**; the

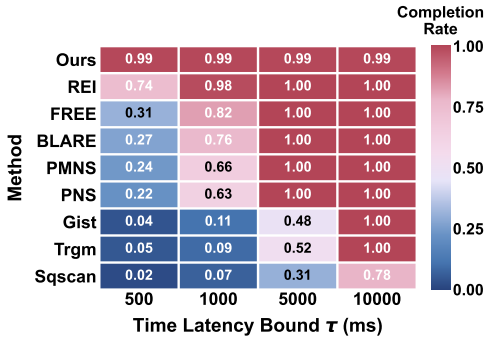



(a) **Backbone trade-off.** Training time vs. accuracy under matched parameter budgets.

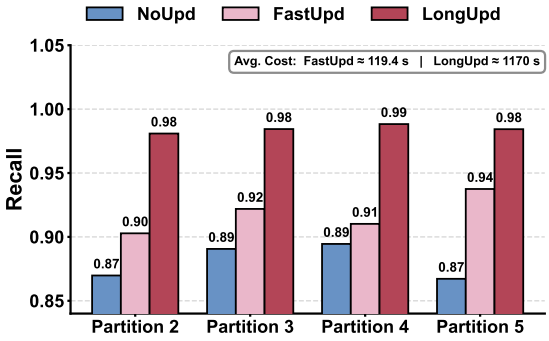



(b) **TPC-H plans.** End-to-end latency for Q2, Q9, Q13, Q14, and Q20 at $k = 64$.

Figure 4: **Model selection and end-to-end query-plan performance.** Left: the Transformer gives the best accuracy–cost operating point. Right: RIVET reduces full TPC-H query-plan latency across all templates.




(a) **Preview completion.** Success@$\tau$ for `LIMIT-`$k$ results within budget $\tau$ (default $k = 64$).




(b) **Workload drift.** Recall@64 under three maintenance policies—NoUpd (no update), FastUpd (lightweight incremental update), and LongUpd (full re-train); labels report average update cost.

Figure 5: **Interactive completion and robustness.** Left: RIVET completes preview retrieval under tight latency budgets. Right: lightweight updates recover most quality under time-ordered workload drift.

full beam-plus-temperature key set reaches **0.985–0.986**. This directly supports the coverage–cost view in Sec. 4.3: performance comes from concentrating probability mass on effective keys and then using diversity to cover multiple candidate regions. Per-dataset ablations are in App. L. Increasing the key budget from 4 to 16 raises Recall@64 from 0.851–0.934 to 0.976–0.991 across datasets; budgets 16–64 reach a quality plateau with under 10% end-to-end latency variation (App. I.5).

**Column specialization and supervision.** Table 1 fixes architecture, decoder, key budget, and test queries. Pooling matches either specialist compute or per-column exposure ($4\times$ training), with average gaps of 7.2 and 2.3 percentage points. At fixed data and training volume, shuffling IMDB regex–prefix pairs reduces Recall@64 by 39.1 points. A non-neural ranker combining bucket cardinality, sampled hit rate, AST compatibility, and estimated yield reaches 0.338 Recall@64 versus RIVET's 0.985 at the same key budget (App. L.1).

▼ **Exp.3. Translator backbone.** Figure 4a compares translator backbones under matched parameter budgets and identical training settings. The Transformer reaches the best accuracy–cost operating point, attaining **0.9886** prefix-key accuracy with **2.34k s** training time. This matches the task structure: regex-to-key translation depends on non-local constraints such as branches, quantifiers, and scope boundaries, where global self-attention gives a cleaner inductive bias than recurrent or convolutional alternatives. Matched parameter counts and training settings are in App. F.

Table 1: Column specialization. Deltas are percentage points relative to the specialist; pooled settings match total compute or per-column exposure.

| Target | Specialist | Best cross-column | Pooled, compute | Pooled, exposure |
|---|---|---|---|---|
| IMDB | .986 | $-21.3$ | $-5.2$ | $-1.6$ |
| WIKI | .985 | $-17.4$ | $-4.3$ | $-1.3$ |
| TPC-H | .985 | $-34.8$ | $-8.7$ | $-2.8$ |
| US-Acc | .985 | $-42.6$ | $-10.4$ | $-3.5$ |

### 5.3 Downstream System Utility

▼ **Exp.4. End-to-end plans.** We integrate RIVET into full TPC-H multi-table query plans using official templates Q2, Q9, Q13, Q14, and Q20. Figure 4b shows that RIVET gives the lowest latency on every query, **18.7–56.7 ms**, and improves over REI by **1.23×–1.77×** (geometric mean ≈**1.38×**). All methods reach the SEQSCAN oracle Recall@64 $= 1$, so the **1.23×–1.77× speedup directly reflects RIVET's lower filtering and verification overhead within full multi-stage plans**.

▼ **Exp.5. Preview retrieval.** Figure 5a tests whether the system can reliably return preview results under tight latency budgets. At $k = 64$ and $\tau = 500$ ms, RIVET completes **99%** of queries, compared with **74%** for REI. For common smaller previews ($k = 16/32$), RIVET reaches **100%** completion across all reported latency budgets; at $k = 128$, RIVET reaches **0.98** completion at 500 ms. Additional preview results are in App. E.4.

▼ **Exp.6. Interactive membership.** We further test an agent-in-the-loop membership workflow: given a regex and a target entity, the system returns top-$k$ matches in rounds until the target appears. On `imdb_name` with $k = 16$, RIVET needs **531 ms** total time, a **3.20×** speedup over REI and **27.0×** over SEQSCAN. It also requires **1.00** rounds on average, while other methods need **1.11–1.35**. This provides an application-facing check that the latency–coverage trade-off carries over to repeated interactive use (App. E.3).

▼ **Exp.7. Deployment cost.** RIVET's one-time training cost is about **2,300 s** (∼39 min) per text column and the FP16 translator is **58–60 MB**. For fixed-budget preview retrieval, amortized against REI on the OOD workload, the break-even point is **0.15 days** on TPC-H, **0.39 days** on US-Acc, **0.77 days** on IMDB, and **1.93 days** on WIKI at 5K queries/day; all four amortize within two days under this workload and execution mode. The shared B-tree, trgm GIN, REI storage, online latency decomposition, and residual-scan costs are detailed in App. I. CPU-only translator inference takes 153.6 ms median and 181.9 ms p90. Remote execution transfers 296 B median and 607 B maximum per request/response and adds one RPC RTT. In the controlled WIKI-W4 row-count sweep, SEQSCAN wins at 5K rows and RIVET at 10K (App. I.4).

### 5.4 Robustness and OOD Behavior

▼ **Exp.8. Real-world regex generalization.** We collect 80 human-authored regexes from StackOverflow Q&A and arXiv documents, with no model retraining; these patterns exhibit higher character-class density, anchor density, and suffix/infix literal usage than W1–W4. With every baseline set to `LIMIT` $h(q)$, where $h(q)$ is RIVET's verified output count on query $q$, RIVET takes **48.26–92.41 ms**; REI takes 286.15–3,105.58 ms and BLARE 910.63–13,471.74 ms (App. H.2). At the original limit of 64 (Table 2), fixed-budget RIVET reaches Recall@64 of 0.622–0.878; residual scanning raises it to 1.000 and is 3.6–33.9× faster than SEQSCAN. The fastest complete path varies by dataset, motivating cost-based selection. Suffix/infix and dispersed-match queries cause the main coverage losses (App. H.4).

**Cold start.** Training needs column records, without historical queries. On held-out WIKI, 100/1K/10K/Full records yield Recall@64 of 0.180/0.345/0.714/0.985; small-data settings fix architecture, training budget, and decoder. 10K records recover 72.5% of Full quality. REI or SEQSCAN serves queries until held-out recall supports activation (App. J.1).

▼ **Exp.9. Workload drift.** Figure 5b evaluates three update policies on the time-ordered `lineitem.comment` stream. Without retraining, NoUpd retains Recall@64 at **0.87–0.89**; **Fas-**

Table 2: OOD coverage and latency at $k = 64$. Each cell reports Recall@64 / ms. Bold latencies mark the fastest measured path with Recall@64 $= 1$.

| Method | IMDB | US-Acc | WIKI | TPC-H |
|---|---|---|---|---|
| RIVET | .866 / 54.37 | .804 / 67.18 | .622 / 48.26 | .878 / 92.41 |
| RIVET + scan | 1 / **1,100.03** | 1 / 3,549.65 | 1 / 1,913.80 | 1 / 4,597.69 |
| REI | .750 / 651.42 | 1 / **1,304.51** | .950 / 285.72 | 1 / **3,232.83** |
| BLARE | 1 / 2,099.84 | 1 / 6,182.62 | 1 / **927.78** | 1 / 14,060.80 |
| Seqscan | 1 / 20,904.40 | 1 / 49,156.60 | 1 / 6,904.69 | 1 / 155,890.07 |

**tUpd at 119 s/partition recovers Recall@64 to 0.90–0.94 with negligible maintenance overhead**; LongUpd at **1,170 s/partition** ($\sim$10$\times$ the update cost) approaches oracle coverage at **0.98–0.99**, so practitioners can pick the operating point that matches their drift budget. Periodic recall audits can trigger an update or route queries to a fallback; the 500-query window and 0.92 threshold are the evaluated maintenance setting. Workload-transfer across W1–W4 and trigger rules are in App. D.2.1 and App. J.

▼ **Exp.10. LLM and pretraining controls.** Column-specific training is decisive over generic language priors. Four zero-shot LLMs (DeepSeek-V3.2 671B DeepSeek-AI [2025] and Qwen3.5 35B/9B/4B Qwen Team [2026]) achieve only **26%–33%** average dictionary-hit rate for generated prefix keys, versus **97%–99%** for RIVET, while their end-to-end inference is **167$\times$–1,040$\times$** slower. Fine-tuning ByT5-Small Xue et al. [2022] reaches only **53.6%** prefix-key accuracy on IMDB under matched supervision, compared with **98.9%** for the 30M-parameter RIVET Transformer, and is about **15$\times$** slower at inference. These controls support the paper's claim that useful keys are data-column-specific rather than general natural-language priors (App. K).

**Takeaways.** The ten experiments isolate one mechanism: decoded keys concentrating mass on effective buckets plus sampling diversity drive the **0.002**$\rightarrow$**0.985** Recall@64 progression in Exp.1–Exp.3, and the same trained translator carries this mechanism unchanged through Exp.4–Exp.10's downstream plans, OOD stress, drift, and language-prior controls.

## 6 Limitations

RIVET is most effective when a regex identifies a few match-rich prefix buckets. Suffix/infix conditions, pure character classes, and dispersed matches can exceed its key budget; residual scanning completes the limit at the measured cost in Sec. 5.4. Each column needs training data and maintenance, quantified by the cold-start, pooled-training, and drift controls.

The main latency results use colocated GPU inference on 2.0M–24.0M-row columns; CPU inference, network RTT, and small-table scan costs affect deployment (Sec. 5.3). The sampling analysis assumes independent proposals, and the expected-cost formula assumes random candidate order. The full decoder and OOD operating points are evaluated empirically.

## 7 Conclusion

RIVET turns regex retrieval over a large text column into column-specific key generation, with a standard database index serving the generated keys and the regex engine verifying every output. The translator learns from anchor records drawn from the target column, so its keys land in candidate buckets that both exist in the index and contain real matches. Across four datasets and four workloads, RIVET keeps Recall@64 above 0.98 in distribution and improves TPC-H query plans and interactive previews. The experiments quantify column specialization, cold-start behavior, and the cost of completing OOD queries through residual scanning. Column-specific key generation connects learned proposals to the indexes and exact verifiers already available in database systems.

## Acknowledgments and Disclosure of Funding

This work is supported by the National Natural Science Foundation of China (NSFC) under Grant Nos. 62232005 and 62202126. The authors declare no competing interests.

## References


Apache Software Foundation. Apache Lucene. https://lucene.apache.org/, 2024. Accessed: 2026-01-26.

Hannah Bast and Marjan Celikik. Efficient fuzzy search in large text collections. *ACM Transactions on Information Systems*, 31(2):1–59, 2013. doi: 10.1145/2457465.2457470.

Angela Bonifati, Stefania Dumbrava, George Fletcher, Jan Hidders, Matthias Hofer, Wim Martens, Filip Murlak, Joshua Shinavier, Sławek Staworko, and Dominik Tomaszuk. Threshold queries in theory and in the wild. *Proceedings of the VLDB Endowment*, 15(5):1105–1118, 2022. doi: 10.14778/3510397.3510407.

Michael J. Carey and Donald Kossmann. On saying "enough already!" in sql. In *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 219–230, 1997. URL https://dblp.org/rec/conf/sigmod/CareyK97.

Michael J. Carey and Donald Kossmann. Reducing the braking distance of an SQL query engine. In *Proceedings of the 24th International Conference on Very Large Data Bases (VLDB)*, pages 158–169, 1998. URL https://dblp.org/rec/conf/vldb/CareyK98.html.

Kevin Chen-Chuan Chang and Seung-won Hwang. Minimal probing: supporting expensive predicates for top-k queries. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 346–357, 2002. doi: 10.1145/564691.564731. URL https://dblp.org/rec/conf/sigmod/ChangH02.

Surajit Chaudhuri and Luis Gravano. Evaluating top-k selection queries. In *Proceedings of the 25th International Conference on Very Large Data Bases (VLDB)*, pages 397–410, 1999. URL https://dblp.org/rec/conf/vldb/ChaudhuriG99.

Junghoo Cho and Sridhar Rajagopalan. A fast regular expression indexing engine. In *Proceedings of the 18th International Conference on Data Engineering (ICDE)*, pages 419–430, 2002. URL https://dblp.org/rec/conf/icde/ChoR02.

Russ Cox. RE2: A fast, safe, thread-friendly regular expression engine. https://github.com/google/re2, 2010. Accessed: 2026-01-26.

DeepSeek-AI. DeepSeek-V3.2: Pushing the frontier of open large language models, 2025. URL https://arxiv.org/abs/2512.02556.

Jialin Ding, Umar Farooq Minhas, Jia Yu, Chi Wang, Jaeyoung Do, Yinan Li, Hantian Zhang, Badrish Chandramouli, Johannes Gehrke, Donald Kossmann, David Lomet, and Tim Kraska. ALEX: An updatable adaptive learned index. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 969–984, 2020. doi: 10.1145/3318464.3389711.

Ronald Fagin, Amnon Lotem, and Moni Naor. Optimal aggregation algorithms for middleware. In *Proceedings of the Twentieth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems (PODS '01)*, pages 102–113. ACM, 2001. doi: 10.1145/375551.375567.

Benjamin Hilprecht, Andreas Schmidt, Moritz Kulessa, Alejandro Molina, Kristian Kersting, and Carsten Binnig. DeepDB: Learn from data, not from queries! *Proceedings of the VLDB Endowment*, 13(7):992–1005, 2020. doi: 10.14778/3384345.3384349.

Ihab F. Ilyas, Walid G. Aref, and Ahmed K. Elmagarmid. Supporting top-k join queries in relational databases. *The VLDB Journal*, 13(3):207–221, 2004. doi: 10.1007/s00778-004-0128-2. URL https://link.springer.com/article/10.1007/s00778-004-0128-2.

Min-Soo Kim, Kyu-Young Whang, Jae-Gil Lee, and Min-Jae Lee. n-Gram/2L: A space and time efficient two-level n-gram inverted index structure. In *Proceedings of the 31st International Conference on Very Large Data Bases (VLDB)*, pages 325–336, 2005. URL https://dl.acm.org/doi/10.5555/1083592.1083632.

Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. The case for learned index structures. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD)*, pages 489–504, 2018. doi: 10.1145/3183713.3196909. URL https://dblp.org/rec/conf/sigmod/KraskaBCDP18.html.

Anirban Majumder, Rajeev Rastogi, and Sriram Vanama. Scalable regular expression matching on data streams. In *Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 161–172, 2008. doi: 10.1145/1376616.1376635.

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, and Theo Vassilakis. Dremel: Interactive analysis of web-scale datasets. *Proceedings of the VLDB Endowment*, 3(1-2):330–339, 2010. doi: 10.14778/1920841.1920886. URL https://dl.acm.org/doi/10.1145/1953122.1953148.

Microsoft. take operator. https://learn.microsoft.com/en-us/kusto/query/take-operator, 2024. Accessed October 19, 2026.

Microsoft. Search content in a review set with the advanced review set explorer (preview). https://learn.microsoft.com/en-us/purview/edisc-review-set-explorer, 2026. Accessed October 19, 2026.

PostgreSQL Global Development Group. pg_trgm — support for similarity of text using trigram matching. PostgreSQL 16 Documentation. https://www.postgresql.org/docs/16/pgtrgm.html, 2024. Accessed: 2026-01-26.

PostgreSQL Global Development Group. 65.2. GiST Indexes. https://www.postgresql.org/docs/current/gist.html, 2025. Accessed: 2025-03-18.

Tao Qiu, Xiaochun Yang, Bin Wang, and Wei Wang. Efficient regular expression matching based on positional inverted index. *IEEE Transactions on Knowledge and Data Engineering*, 34(3): 1133–1148, 2022. doi: 10.1109/TKDE.2020.2992295.

Qwen Team. Qwen3.5: Model cards and official release. Hugging Face Collection. https://huggingface.co/collections/Qwen/qwen35, 2026. Models include Qwen3.5-35B-A3B, Qwen3.5-9B, Qwen3.5-4B. Accessed: 2026-05-07.

Cristian Riveros, Nicolás Van Sint Jan, and Domagoj Vrgoč. Rematch: a novel regex engine for finding all matches. *Proceedings of the VLDB Endowment (PVLDB)*, 16:2792–2804, 2023. URL https://www.vldb.org/pvldb/vol16/p2792-vrgoc.pdf.

Vivek Sharma. How to filter wikipedia petscan API results using regular expressions? https://stackoverflow.com/questions/60224177/, 2020.

Suraj Shetiya, Saravanan Thirumuruganathan, Nick Koudas, and Gautam Das. Astrid: Accurate selectivity estimation for string predicates using deep learning. *Proceedings of the VLDB Endowment*, 14(4):471–484, 2020. doi: 10.14778/3436905.3436907. URL https://www.cse.iitb.ac.in/~surajs/assets/pdf/shetiya2020astrid.pdf.

Xiang Wang, Yang Hong, Harry Chang, KyoungSoo Park, Geoff Langdale, Jiayu Hu, and Heqing Zhu. Hyperscan: A fast multi-pattern regex matcher for modern CPUs. In *16th USENIX Symposium on Networked Systems Design and Implementation (NSDI 19)*, pages 631–648, 2019.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. ByT5: Towards a token-free future with pre-trained byte-to-byte models. *Transactions of the Association for Computational Linguistics*, 10:291–306, 2022. doi: 10.1162/tacl_a_00461. URL https://aclanthology.org/2022.tacl-1.17.

Xiaochun Yang, Bin Wang, and Chen Li. 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 (SIGMOD)*, pages 353–364, 2008. doi: 10.1145/1376616.1376655.

Xiaochun Yang, Bin Wang, Tao Qiu, Yaoshu Wang, and Chen Li. Improving regular-expression matching on strings using negative factors. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD)*, pages 361–372, 2013. doi: 10.1145/2463676.2465289. URL https://dblp.org/rec/conf/sigmod/YangWQWL13.

Xiaochun Yang, Tao Qiu, Bin Wang, Baihua Zheng, Yaoshu Wang, and Chen Li. Negative factor: Improving regular-expression matching in strings. *ACM Transactions on Database Systems*, 40(4): 1–46, 2016. doi: 10.1145/2847525.

Zongheng Yang, Amog Kamsetty, Sifei Luan, Eric Liang, Yan Duan, Xi Chen, and Ion Stoica. Neurocard: One cardinality estimator for all tables. *Proceedings of the VLDB Endowment*, 14(1): 61–73, 2020. doi: 10.14778/3421424.3421432. URL https://vldb.org/pvldb/vol14/p61-yang.pdf.

Ling Zhang, Shaleen Deep, Avrilia Floratou, Anja Gruenheid, Jignesh M. Patel, and Yiwen Zhu. Exploiting structure in regular expression queries. *Proceedings of the ACM on Management of Data*, 1(2):1–28, 2023. doi: 10.1145/3589297.

Ling Zhang, Shaleen Deep, Jignesh M. Patel, and Karthikeyan Sankaralingam. Regular expression indexing for log analysis. *Proceedings of the ACM on Management of Data*, 3(6):1–29, 2025. doi: 10.1145/3769820.

# A Formalization of LIMIT-$k$ Regex Retrieval

## A.1 Matching and retrieval objective

Let $D = \{t_i\}_{i=1}^n \subseteq \Sigma^*$ be a text column and $R(Q)$ the language of regex $Q$, formed by concatenation, union, and Kleene closure. The regex appears in $t$ when some substring $t[a, b)$ belongs to $R(Q)$. Write $W(Q) = \{t \in D : Q \text{ appears in } t\}$. Exact unordered LIMIT-$k$ retrieval returns any $S \subseteq W(Q)$ with $|S| = \min\{k, |W(Q)|\}$. A budgeted execution returns a verified set $\widehat{S}(Q) \subseteq W(Q)$ of size at most this target. Every output passes the native regex verifier, so precision is 1. We measure $\mathrm{cost}(Q; k)$ by the number of regex verifications issued before reaching $k$ matches, candidate exhaustion, or an external budget.

## A.2 Keys, buckets, and online output

For a fixed-length prefix extractor $\phi$, define

$$\mathcal{K}(D) = \{\phi(t) : t \in D\}, \qquad \mathrm{Cand}(p) = \{t \in D : \phi(t) = p\}, \qquad \mathrm{Cand}(K) = \bigcup_{p \in K} \mathrm{Cand}(p).$$

Each record has one key, so distinct buckets are disjoint. Write $c(p) = |\mathrm{Cand}(p)|$ and $h(p; Q) = |\mathrm{Cand}(p) \cap W(Q)|$; bucket sizes and hit counts therefore add over a key set. The translator defines $q_{\theta,T}(p \mid Q)$ at temperature $T > 0$, and the retained set obeys $K_\theta(Q) \subseteq \mathcal{K}(D)$, $|K_\theta(Q)| \le t_{\max}$. Its verified output satisfies

$$\widehat{S}(Q) \subseteq \mathrm{Cand}(K_\theta(Q)) \cap W(Q), \qquad |\widehat{S}(Q)| \le k, \qquad \mathrm{cost}(Q; k) \le |\mathrm{Cand}(K_\theta(Q))|.$$

# B Online Execution and Translator Details

## B.1 Key generation and fixed-draw protocol

The deployed decoder combines beam proposals and temperature samples. It checks proposals against $\mathcal{K}(D)$, skips duplicates, and retains distinct keys in proposal order until $t_{\max}$ is reached. The stochastic analysis instead fixes $S$ independent draws from $q_{\theta,T}$ before truncation: out-of-dictionary samples are discarded without resampling, and deduplication yields $\Pi_\theta(Q)$. App. C analyzes this sampled set; App. L measures the deployed decoder variants.

## B.2 Residual scanning for a filled limit

If the key path returns fewer than $k$ verified records and the application requests a filled limit, scanning continues over the remaining data with the original regex. Record identifiers prevent duplicate output. Execution ends at $k$ distinct matches or column exhaustion, returning $\min\{k, |W(Q)|\}$ records. Table 2 includes both the initial key path and residual scan in its end-to-end timing.

## B.3 Character-level translator

The Transformer encoder–decoder emits a length-$\ell$ prefix autoregressively as specified in Sec. 4.3. At each position, temperature scales the logits before softmax. Teacher forcing supplies the ground-truth prefix history and maximizes $\log q_{\theta,1}(p \mid Q)$. Character-level tokenization preserves regex operators and escaped literals. Matched backbone architectures and training recipes are in App. F.

# C Coverage–Cost Analysis

## C.1 Fixed-draw inclusion and coverage

For $S$ independent draws $p^{(i)} \sim q_{\theta,T}(\cdot \mid Q)$, dictionary filtering and deduplication give the key set $\Pi_\theta(Q)$. A dictionary key appears exactly when it is drawn at least once, hence

$$\rho_\theta(p; Q) := \Pr[p \in \Pi_\theta(Q)] = 1 - (1 - q_{\theta,T}(p \mid Q))^S. \tag{2}$$

Disjoint buckets and linearity of expectation give

$$\begin{aligned}\mathbb{E}[|\mathrm{Cand}(\Pi_\theta(Q))|] &= \sum_{p\in\mathcal{K}(D)} c(p)\rho_\theta(p;Q),\\ \mathbb{E}[|\mathrm{Cand}(\Pi_\theta(Q))\cap W(Q)|] &= \sum_{p\in\mathcal{K}(D)} h(p;Q)\rho_\theta(p;Q).\end{aligned} \tag{3}$$

Increasing inclusion probability raises coverage and can also increase the candidate volume sent to verification.

### C.2 Probability of hitting an effective key

Let $\mathcal{K}^+(Q) = \{p \in \mathcal{K}(D) : h(p;Q) > 0\}$ and $\mu_\theta(Q) = \sum_{p\in\mathcal{K}^+(Q)} q_{\theta,T}(p \mid Q)$.

**Proposition C.1.** *Under i.i.d. fixed draws, the probability that $\Pi_\theta(Q)$ contains an effective key is given by Eq.* (1)*.*

**Proof.** Each draw misses $\mathcal{K}^+(Q)$ with probability $1-\mu_\theta(Q)$. Independence gives miss probability $(1-\mu_\theta(Q))^S$ for all draws; taking the complement proves the result. □

### C.3 Anchor-Sampling Identifiability

We formalize the supervision distribution induced by the anchor-based generator described in App. D.2, and show that the population optimum of cross-entropy training is supported on $\mathcal{K}^+(Q)$, with explicit weights in the uniform-anchor case.

For a regex $Q$, let $\nu_Q$ be the anchor distribution over records used to synthesize training pairs $(Q,\, p = \phi(t))$. By construction, anchors satisfy $\mathrm{supp}(\nu_Q) \subseteq D \cap W(Q)$: only matching records are sampled, because the generator instantiates $Q$ from $t$ itself. The induced marginal on keys is

$$P(p \mid Q) := \Pr_{t\sim\nu_Q}[\phi(t) = p]\,. \tag{4}$$

**Connection to the actual generator.** The generator described in App. D.2 produces training pairs by sampling a record $t$ and synthesizing a regex $Q$ from $t$ such that $t \in W(Q)$ holds by construction. Writing the joint distribution as $\Pr(t, Q)$, the conditional anchor distribution is

$$\nu_Q(t) := \Pr(t \mid Q), \qquad \mathrm{supp}(\nu_Q) \subseteq D \cap W(Q), \tag{5}$$

because the generator emits $Q$ only from anchors that match it. The marginal in Eq. (4) is therefore well-defined and satisfies the support hypothesis of Proposition C.2 for any generator obeying the anchor-match invariant. The uniform-anchor special case in Proposition C.2(2) holds whenever $\Pr(t \mid Q)$ is constant on $\mathrm{supp}(\nu_Q)$.

**Proposition C.2.** *Under any anchor distribution $\nu_Q$ with $\mathrm{supp}(\nu_Q) \subseteq D \cap W(Q)$, the marginal $P(\cdot \mid Q)$ in Eq.* (4) *satisfies:*

1. $\mathrm{supp}(P(\cdot \mid Q)) \subseteq \mathcal{K}^+(Q)$*;*
2. *if $\nu_Q$ is uniform on $D\cap W(Q)$, then $P(p \mid Q) = h(p;Q)/|D\cap W(Q)|$ for every $p \in \mathcal{K}^+(Q)$.*

**Proof.** For (1): $\phi(t) = p$ implies $t \in \mathrm{Cand}(p)$, and $t \in W(Q)$ by $\mathrm{supp}(\nu_Q) \subseteq W(Q)$, hence $t \in \mathrm{Cand}(p) \cap W(Q)$, which forces $h(p;Q) \geq 1$ and therefore $p \in \mathcal{K}^+(Q)$. For (2): under uniform $\nu_Q$, $P(p \mid Q) = |\{t \in D \cap W(Q) : \phi(t) = p\}|/|D \cap W(Q)| = h(p;Q)/|D \cap W(Q)|$ by the definitions of $h$ and $\phi$. □

**Bayes-optimal supervision.** Let $\mathcal{L}(\theta) := \mathbb{E}_Q\big[\,\mathbb{E}_{t\sim\nu_Q}[-\log q_{\theta,1}(\phi(t) \mid Q)]\,\big]$ denote the population cross-entropy. Minimizing $\mathcal{L}$ over the space of *all* conditional distributions on $\mathcal{K}(D)$ yields $q^\star(\cdot \mid Q) = P(\cdot \mid Q)$, so by Proposition C.2(1),

$$\mu^\star_\theta(Q) = \sum_{p\in\mathcal{K}^+(Q)} P(p \mid Q) = 1. \tag{6}$$

When $\nu_Q$ is uniform on $D \cap W(Q)$, Proposition C.2(2) further gives $q^\star(p \mid Q) = h(p;Q)/|D \cap W(Q)|$, so the population-optimal translator weights each prefix by its true-hit count.

**Realizability.** The minimization above is over all conditional distributions on $\mathcal{K}(D)$. If the translator class $\{q_{\theta,1}(\cdot \mid Q) : \theta \in \Theta\}$ contains $P(\cdot \mid Q)$, the parametric argmin coincides with $q^\star$ and $\mu_\theta(Q) \to \mu^\star_\theta(Q) = 1$ at the parametric optimum. Otherwise the parametric argmin is the information projection $q^{\mathrm{proj}}(\cdot \mid Q) := \arg\min_{q_\theta} \mathrm{KL}(P(\cdot \mid Q) \,\|\, q_{\theta,1}(\cdot \mid Q))$, and Proposition C.5 controls $|\mu^\star_\theta(Q) - \mu_\theta(Q)|$ through $\mathrm{KL}(P \,\|\, q_{\theta,1})$ rather than zero. Character-level sequence-to-sequence models with sufficient depth and width are universal approximators of conditional distributions over fixed-length strings.

**Repeated regex hits as multi-positive supervision.** The generator yields the same regex $Q$ across multiple anchors $t_1, \ldots, t_m \sim \nu_Q$, producing pairs $(Q, \phi(t_1)), \ldots, (Q, \phi(t_m))$ with possibly distinct keys. Empirical risk on these pairs is an unbiased estimator of $\mathcal{L}(\theta)$, so the supervision generated by repeated hits converges to the Bayes optimum identified above as the number of anchors per regex grows.

### C.4 Verification-Cost Bound via Negative-Hypergeometric Sampling

Proposition C.2 controls how training shapes the key distribution. We now connect that distribution to the system-level quantity $\mathrm{cost}(Q; k)$ defined in App. A.1.

For an online key set $K \subseteq \mathcal{K}(D)$, write

$$H(K) := \sum_{p \in K} h(p; Q), \qquad B(K) := \sum_{p \in K} c(p), \tag{7}$$

the total true hits and total candidates pooled over the keys in $K$. By disjointness of $\{\mathrm{Cand}(p)\}_p$, the union $\bigcup_{p \in K} \mathrm{Cand}(p)$ has cardinality $B(K)$ and contains exactly $H(K)$ records that match $Q$.

**Theorem C.3.** *Fix a query $Q$, an online key set $K = K_\theta(Q)$, and a target $k \geq 1$. Suppose the executor verifies records in $\bigcup_{p \in K} \mathrm{Cand}(p)$ in a uniformly random order, stopping when $k$ matches have been confirmed or when the candidate pool is exhausted. Then*

1. *(Unconditional bound)* $\mathrm{cost}(Q; k) \leq B(K)$ *deterministically.*
2. *(Conditional expectation) On the event* $\{H(K) \geq k\}$,

$$\mathbb{E}\big[\, \mathrm{cost}(Q; k) \,\big|\, H(K) \geq k \,\big] \;\leq\; \frac{k\,(B(K)+1)}{H(K)+1}. \tag{8}$$

**Proof.** (1) The executor terminates no later than candidate exhaustion, so at most $B(K)$ verifications are issued. (2) Under uniformly random order, the index of the $k$-th match in a population of $B(K)$ records containing $H(K)$ marked items follows a negative-hypergeometric distribution with mean $k(B(K)+1)/(H(K)+1)$ when $H(K) \geq k$; $\mathrm{cost}(Q; k)$ equals this index by definition (verifications until $k$ matches), giving the conditional expectation. □

**Coverage and verification cost.** For independent proposals before truncation, Eq. (3) gives the expected true-hit count, while Eq. (1) gives the probability of reaching at least one effective key. Filling the limit depends on the total hits $H(K)$ in the retained buckets. Conditional on $H(K) \geq k$, Eq. (8) relates the expected verification work to both candidate volume $B(K)$ and hit count $H(K)$.

**Remark on verification order.** The unconditional bound in (1) is order-free. Eq. (8) uses uniform random order to obtain the closed-form negative-hypergeometric mean. When $H(K) \geq k$, under any specific ordering, $\mathrm{cost}(Q; k)$ lies in the interval $[k,\ B(K) - H(K) + k]$: the lower extreme is attained when matches are verified first and the upper extreme when they are verified last; the uniform-order bound is the average over all orderings. Deployments with reliable bucket-yield estimates can verify high-yield buckets first and reduce cost below this average.

**Deployment order.** The executor follows index and key order. The negative-hypergeometric expression averages over uniform permutations of the entire candidate pool; this order model is distinct from index traversal. App. C.8 compares mean costs over 200 random permutations per query with the expression on 97 eligible queries and reports agreement within 5%.

**Concentration of cost.** The exact negative-hypergeometric variance

$$\mathrm{Var}\big[\,\mathrm{cost}(Q;k)\,\big|\,H(K)\geq k\,\big]=\frac{k\,(B(K)+1)(B(K)-H(K))(H(K)-k+1)}{(H(K)+1)^2(H(K)+2)}$$

is bounded by $\mathbb{E}[\mathrm{cost}(Q;k)\mid H(K)\geq k]^2/k$, since $B(K)/(H(K)+2)\leq(B(K)+1)/(H(K)+1)$. Chebyshev's inequality therefore gives, for every $\lambda>1$,

$$\Pr\left[\,\mathrm{cost}(Q;k)>\lambda\cdot\frac{k\,(B(K)+1)}{H(K)+1}\;\middle|\;H(K)\geq k\right]\;\leq\;\frac{1}{k\,(\lambda-1)^2}. \tag{9}$$

The high-probability bound is independent of $B(K)$ and $H(K)$ and sharpens with $k$, so the values $k\in\{16,32,64,128\}$ used in our experiments give substantial concentration around the negative-hypergeometric mean.

### C.5 Schur-Concavity of Effective-Key Coverage

Theorem C.3 shows that the verification cost decreases when $H(K)$ grows. Eq. (3) expresses $\mathbb{E}[H(K)]$ through inclusion probabilities $\rho_\theta$, which depend on $q_{\theta,T}$ only through the per-key probabilities $q_{\theta,T}(p\mid Q)$. Given a fixed total mass $\mu_\theta(Q)$ on $\mathcal{K}^+(Q)$, expected effective-key coverage is maximized when this mass is spread across multiple effective keys and minimized when concentrated on one.

For a probability vector $\mathbf{q}=(q_p)_{p\in\mathcal{K}^+(Q)}$ with $\sum_p q_p=\mu_\theta(Q)$, define the expected effective-key coverage

$$C_S(\mathbf{q}):=\sum_{p\in\mathcal{K}^+(Q)}\big(1-(1-q_p)^S\big). \tag{10}$$

$C_S(\mathbf{q})=\mathbb{E}[\,|\Pi_\theta(Q)\cap\mathcal{K}^+(Q)|\,]$ when $\mathbf{q}$ is the restriction of $q_{\theta,T}(\cdot\mid Q)$ to $\mathcal{K}^+(Q)$.

**Proposition C.4.** *The map $\mathbf{q}\mapsto C_S(\mathbf{q})$ is Schur-concave on the simplex $\Delta_{\mathcal{K}^+(Q)}$. Consequently, for fixed $\mu_\theta(Q)$,*

$$C_S(\mathbf{q})\geq 1-(1-\mu_\theta(Q))^S \qquad \textit{(attained when } \mathbf{q} \textit{ concentrates on one } p^\star\textit{);} \tag{11}$$

$$C_S(\mathbf{q})\leq|\mathcal{K}^+(Q)|\Big(1-\big(1-\tfrac{\mu_\theta(Q)}{|\mathcal{K}^+(Q)|}\big)^S\Big) \qquad \textit{(attained when } \mathbf{q} \textit{ is uniform on } \mathcal{K}^+(Q)\textit{).} \tag{12}$$

**Proof.** The scalar function $f(x):=1-(1-x)^S$ has second derivative $f''(x)=-S(S-1)(1-x)^{S-2}\leq 0$ on $[0,1]$ for $S\geq 1$, so $f$ is concave. $C_S$ is the sum of $f$ applied coordinate-wise to $\mathbf{q}$, and sums of identical concave functions are Schur-concave on the simplex by the Hardy–Littlewood–Pólya majorization theorem. The two bounds correspond to the two extremes of the majorization order on the level set $\{\mathbf{q}:\sum_p q_p=\mu_\theta(Q)\}$: full concentration majorizes every distribution with the same total mass and minimizes a Schur-concave function, while uniformity is majorized by every such distribution and maximizes it. □

**Expected hits before truncation.** For the sampled set $\Pi_\theta(Q)$ and a nonempty $\mathcal{K}^+(Q)$, let $h_{\min}(Q)=\min_{p\in\mathcal{K}^+(Q)}h(p;Q)$. Then

$$\mathbb{E}[H(\Pi_\theta(Q))]\geq h_{\min}(Q)\sum_{p\in\mathcal{K}^+(Q)}\rho_\theta(p;Q)\geq h_{\min}(Q)\big(1-(1-\mu_\theta(Q))^S\big). \tag{13}$$

The second inequality follows from Eq. (11).

**Implication for decoder choice.** Decoders that draw $S$ i.i.d. samples from $q_{\theta,T}(\cdot\mid Q)$ — the stochastic protocol of App. B.1 — achieve expected effective-key coverage exactly $C_S(\mathbf{q})$, which Proposition C.4 bounds between Eq. (11) and Eq. (12). Greedy decoding instead returns the single argmax of $q_{\theta,T}$ deterministically and achieves binary coverage $\mathbf{1}\{\arg\max_p q_{\theta,T}(p\mid Q)\in\mathcal{K}^+(Q)\}\in\{0,1\}$, regardless of $S$. Sampling reaches more effective keys in expectation when $C_S(\mathbf{q})>1$. The size of this gain depends on the sampling budget and the distribution of probability mass across effective keys.

**Algorithm–theory alignment.** Proposition C.4 describes independent temperature samples before truncation. The deployed decoder also includes beam proposals and caps the combined key set. App. L measures these choices: beam-only decoding reaches Recall@64 of 0.945–0.961, and the full key set reaches 0.985–0.986.

### C.6 From Cross-Entropy Training to Coverage Gap

Proposition C.2 identifies the population optimum of the supervised cross-entropy loss; Theorem C.3 relates candidate hits and volume to verification cost. The remaining link is between the training-time CE loss and the online coverage mass $\mu_\theta(Q)$ achieved by the trained model.

For each $Q$, write $P_Q := P(\cdot \mid Q)$ for the Bayes-optimal distribution from Eq. (4) and $q_Q := q_{\theta,1}(\cdot \mid Q)$ for the trained translator at the training temperature $T = 1$.

**Proposition C.5.** *For every regex $Q$,*

$$\left|\mu^\star_\theta(Q) - \mu_\theta(Q)\right| \;\le\; \|P_Q - q_Q\|_{\mathrm{TV}} \;\le\; \sqrt{\tfrac{1}{2}\,\mathrm{KL}(P_Q \,\|\, q_Q)}. \tag{14}$$

**Proof.** $\mu^\star_\theta(Q) - \mu_\theta(Q) = P_Q(\mathcal{K}^+(Q)) - q_Q(\mathcal{K}^+(Q))$, which is bounded in absolute value by $\sup_{A \subseteq \mathcal{K}(D)} |P_Q(A) - q_Q(A)| = \|P_Q - q_Q\|_{\mathrm{TV}}$. Pinsker's inequality gives $\|P_Q - q_Q\|_{\mathrm{TV}} \le \sqrt{\frac{1}{2}\,\mathrm{KL}(P_Q \| q_Q)}$. □

**Connection to the training risk.** The population cross-entropy decomposes as $\mathcal{L}(\theta) = \mathbb{E}_Q[H(P_Q)] + \mathbb{E}_Q[\mathrm{KL}(P_Q \| q_Q)]$, where $H(P_Q)$ is the entropy of $P_Q$. Minimizing $\mathcal{L}(\theta)$ is therefore equivalent to minimizing $\mathbb{E}_Q[\mathrm{KL}(P_Q \| q_Q)]$, which by Eq. (14) controls the average coverage gap.

**Connection to execution.** The excess population cross-entropy controls the average KL divergence and, through Pinsker's inequality, the gap in effective-key probability mass. Independent sampling converts that mass into a hit probability and an expected hit-count lower bound (Eq. (13)). For a retained key set with $H(K) \ge k$, Theorem C.3 then expresses expected verification cost in terms of $B(K)$ and $H(K)$ under random order. This separates the learning objective, sampled coverage, and the candidate-volume cost of execution.

### C.7 Selective Routing with Record Prefixes

The index key $\phi(t)$ is the prefix of the complete record $t$. We call $\pi \in \Sigma^\ell$ a **forced record prefix** of $Q$ if every record string matched by $Q$, under the executor's matching semantics, starts with $\pi$. A start-of-record assertion followed by a case-sensitive literal of length at least $\ell$ supplies such a prefix.

**Definition C.6.** Fix a column $D$, key length $\ell$, and selectivity threshold $\alpha \in (0, 1]$. We say that $Q$ is $(\ell, \alpha)$*-prefix-routable on* $D$ if there exists $\pi \in \Sigma^\ell$ such that

1. $\pi$ is a forced record prefix of $Q$;
2. $c(\pi) > 0$, i.e., $\mathrm{Cand}(\pi)$ is non-empty in $D$;
3. $c(\pi)/|D| \le \alpha$.

We write $\mathrm{ROUTE}^D_{\ell,\alpha}(Q) = 1$ when this holds, and $\mathrm{ROUTE}^D_{\ell,\alpha}(Q) = 0$ otherwise.

**Checking the sufficient condition.** An AST pass can recognize a start-of-record assertion followed by a case-sensitive literal of length at least $\ell$, retaining the assertion and its matching mode. The literal determines $\pi$; the column's bucket count $c(\pi)$ then determines whether the selectivity threshold holds. The supervision generator in App. D.2 provides the separate anchor-match invariant $t \in W(Q)$, which is the condition used in Proposition C.2.

**Theorem C.7.** *Let* $\mathrm{ROUTE}^D_{\ell,\alpha}(Q) = 1$ *with witness* $\pi$*. Then under any anchor distribution* $\nu_Q$ *with* $\mathrm{supp}(\nu_Q) \subseteq D \cap W(Q)$*, the Bayes-optimal translator from Proposition C.2 satisfies*

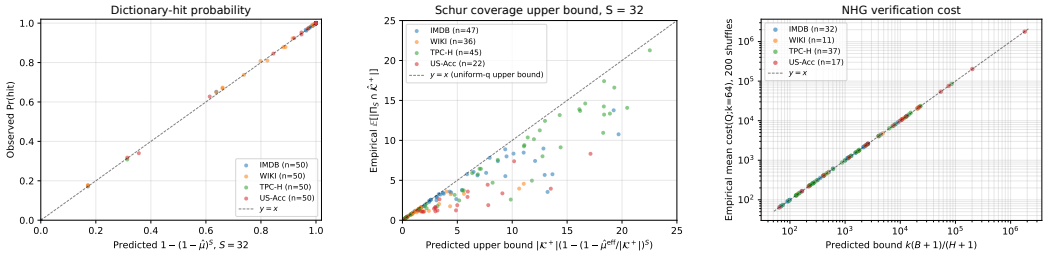
$$W(Q) \subseteq \mathrm{Cand}(\pi), \qquad c(\pi) \le \alpha|D|, \qquad P(\pi \mid Q) = 1. \tag{15}$$

**Proof.** Every matching record starts with $\pi$, so $\phi(t) = \pi$ for every $t \in W(Q)$, proving the inclusion. The bound on $c(\pi)$ is the selectivity condition in Definition C.6. Since $\nu_Q$ is supported on $W(Q)$, Eq. (4) gives $P(\pi \mid Q) = \Pr_{t \sim \nu_Q}[\phi(t) = \pi] = 1$. □

**Operational interpretation.** Retrieving the witness key $\pi$ and verifying its bucket completes exact `LIMIT-`$k$ retrieval within $c(\pi) \le \alpha|D|$ verifications. A start-of-record alternation or a fixed-length character-class sequence can instead yield a finite set of length-$\ell$ prefixes. Its candidate volume is the sum of the disjoint bucket counts, so a small prefix set can also provide selective access. Learned keys additionally exploit column-specific regularities, as measured in Sec. 5.4.

### C.8 Empirical Validation of the Surrogate Chain

We validate the three load-bearing results in App. C.3–C.6 on the four single-column workloads of Sec. 5.1. All experiments use the same trained translators that produce Fig. 3; 50 evaluation queries per column are drawn from the in-distribution workloads of App. D.2.




(a) Dictionary-hit probability (b) Schur upper bound (Prop. C.4) (c) NHG cost (Theorem C.3)

Figure 6: **Empirical validation of the surrogate chain.** Each point is one query (50 per column $\times$ 4 columns). Predictions and observations are produced by the same trained translator used in Fig. 3; tables 3–5 give the per-column counts.

**Dictionary-hit probability.** For each query we draw $N = 512$ i.i.d. temperature samples ($T = 0.5$) and estimate the dictionary-hit probability $\mu_D(Q) = \sum_{p \in \mathcal{K}(D)} q_{\theta,T}(p \mid Q)$. We compare $1 - (1 - \hat{\mu}_D)^S$ with the fraction of size-$S$ sub-blocks containing at least one dictionary key (Table 3). This checks the fixed-draw inclusion calculation at the dictionary-filtering stage; effective-key coverage is measured separately below. The closed-form formula matches the empirical hit rate with per-column mean absolute errors from $0.0001$ to $0.0021$; the per-query Pearson correlation is at least $0.9997$ on every column.

Table 3: **Dictionary-hit probability calibration.** 50 queries per column, $N = 512$ samples per query, $S = 32$ sub-block size. $|\text{err}|$ is the per-query mean of $|\text{predicted} - \text{observed}|$; $\rho$ is the per-query Pearson correlation.

| Dataset | $\bar{\hat{\mu}}_D$ | predicted ($S$=32) | observed ($S$=32) | $\lvert\text{err}\rvert$ | $\rho$ |
|---|---|---|---|---|---|
| IMDB | 0.672 | 0.953 | 0.953 | 0.0006 | 0.9999 |
| WIKI | 0.443 | 0.926 | 0.926 | 0.0021 | 0.9998 |
| TPC-H | 0.824 | 0.986 | 0.986 | 0.0001 | 1.0000 |
| US-Accident | 0.769 | 0.958 | 0.958 | 0.0010 | 0.9997 |

**Prop. C.4: Schur-concave coverage of $\mathcal{K}^+$.** For each query we sample $N = 512$ keys, then probe PostgreSQL with `LIKE 'p%' AND col ~ Q` to identify which sampled prefixes are effective ($h(p; Q) > 0$). Setting $\hat{\mu}_\theta^{\text{eff}} = \sum_{p \in \hat{\mathcal{K}}^+} \hat{q}_\theta(p)$, we compute the concentrated lower bound $1 - (1 - \hat{\mu}_\theta^{\text{eff}})^S$ and the uniform upper bound $|\hat{\mathcal{K}}^+| \left(1 - (1 - \hat{\mu}_\theta^{\text{eff}}/|\hat{\mathcal{K}}^+|)^S\right)$, and compare to bootstrap estimates of $\mathbb{E}[|\Pi_S \cap \hat{\mathcal{K}}^+|]$ (Table 4). On 150 queries with $|\hat{\mathcal{K}}^+| > 1$ (the non-trivial regime), the empirical coverage lies inside the Schur interval on $132/150$ queries at $S = 32$ and $136/150$ at $S = 64$; the dataset means are uniformly bracketed by the two bounds at every $S$. The 18 violations at $S = 32$ are concentrated on small $|\hat{\mathcal{K}}^+|$ (16 of 18 with $|\hat{\mathcal{K}}^+| \leq 5$), and 16 of 18 exceed the upper bound by tiny margins (median $0.005$, max $0.036$), consistent with bootstrap noise in the saturation regime where empirical $|\Pi_S \cap \hat{\mathcal{K}}^+|$ approaches the closed-form upper bound.

**Theorem C.3: NHG cost bound under uniform random verification.** For each query we decode $K_\theta(Q)$ with $t_{\max} = 32$, query PostgreSQL once per key for $(c(p), h(p; Q))$, aggregate to $(B(K), H(K))$, and simulate $\text{cost}(Q; k = 64)$ over 200 uniform shuffles of the pooled candidate list. Table 5 reports the ratio of empirical mean cost to the theoretical bound $k(B + 1)/(H + 1)$ for queries satisfying $H(K) \geq k$. The empirical cost is at the predicted bound within $5\%$ on every query across all four columns, with mean ratio between $0.999$ and $1.003$ per column.

Table 4: **Schur-concave coverage (Prop. C.4).** Counts are queries (with $|\hat{\mathcal{K}}^+| > 1$) for which the empirical $\mathbb{E}[|\Pi_S \cap \hat{\mathcal{K}}^+|]$ lies in $[\,1-(1-\hat{\mu}^{\text{eff}})^S,\ |\hat{\mathcal{K}}^+|(1-(1-\hat{\mu}^{\text{eff}}/|\hat{\mathcal{K}}^+|)^S)\,]$.

| Dataset | $\lvert\hat{\mathcal{K}}^+\rvert>1$ | $\bar{\hat{\mu}}^{\text{eff}}$ | mean $\lvert\hat{\mathcal{K}}^+\rvert$ | $S$=16 in | $S$=32 in | $S$=64 in |
|---|---|---|---|---|---|---|
| IMDB | 47/50 | 0.198 | 28.4 | 37/47 | 40/47 | 41/47 |
| WIKI | 36/50 | 0.075 | 5.1 | 25/36 | 29/36 | 31/36 |
| TPC-H | 45/50 | 0.306 | 47.5 | 43/45 | 43/45 | 44/45 |
| US-Accident | 22/49 | 0.223 | 3.8 | 20/22 | 20/22 | 20/22 |
| **Total** | 150/199 | — | — | **125/150** | **132/150** | **136/150** |

Table 5: **NHG verification-cost bound (Theorem C.3, uniform random order).** Empirical mean cost (200 shuffles) divided by predicted bound $k(B+1)/(H+1)$ on queries with $H(K) \geq k = 64$. Counts in the "in bound" column are queries with empirical mean within $5\%$ of the prediction.

| Dataset | #Queries | $H(K) \geq k$ | in bound | mean ratio | 95th percentile |
|---|---|---|---|---|---|
| IMDB | 50 | 32 | 32/32 | 1.003 | 1.014 |
| WIKI | 50 | 11 | 11/11 | 0.999 | 1.005 |
| TPC-H | 50 | 37 | 37/37 | 1.003 | 1.015 |
| US-Accident | 49 | 17 | 17/17 | 1.003 | 1.014 |
| **Total** | 199 | 97 | **97/97** | — | — |

**Leading-literal structure on the OOD workload.** The empirical AST feature $A(Q)$ tests for a leading contiguous literal of length $\geq 3$ after stripping zero-width assertions, as implemented in the workload analysis. On the 80 OOD queries of Sec. 5.4, $56/80$ have $A(Q) = 0$, including *all* 13 *queries with* $\text{Recall@64} < 0.5$. Other queries in this group achieve high recall through column-specific regularities captured by the translator. This association makes leading-literal structure a useful query feature alongside the column bucket statistics used for routing.

# D Experimental Protocols

## D.1 Data and workload construction

We use WIKI titles, IMDB names, TPC-H `lineitem.comment`, and US-Accident descriptions. Table 6 reports the cleaned columns after removing NULL and empty strings.

Table 6: Dataset statistics for the text columns used in our evaluation. Length is measured in characters after removing NULL/empty strings.

| **Dataset** | **Total** | **Unique** | **Unique(%)** | **Min** | **Max** | **Avg** |
|---|---|---|---|---|---|---|
| WIKI | 2,000,000 | 2,000,000 | 100.00 | 1 | 251 | 19.83 |
| IMDB | 5,000,000 | 4,210,409 | 84.21 | 1 | 95 | 13.52 |
| US-Accident | 7,728,389 | 3,761,578 | 48.67 | 2 | 679 | 68.84 |
| TPC-H | 23,996,604 | 15,813,794 | 65.90 | 10 | 43 | 26.50 |

## D.2 Workload generation

The same PostgreSQL-compatible generator supplies training pairs and evaluation workloads. It samples an anchor record $t$, selects up to $k_{\max}$ non-punctuation tokens, and extracts token-derived islands in their original order. Metacharacters are escaped and word boundaries use `\y`; digit runs may be generalized to `[0-9]{k}`. Alphabetic islands use sampled $n$-gram substrings. Consecutive islands are connected by bounded gaps $G_j = \texttt{(?:.|\n)}^{\{0,N_j\}}?$, where $N_j = \min(gap_cap, gap_base + gap_per_token \cdot dist_j)$. The constructed regex matches its anchor.

Training pairs are $(Q, \phi(t))$. Stochastic anchor and island selection permit repeated regexes with different anchor prefixes, giving multi-positive supervision under $\nu_Q(t) = \Pr(t \mid Q)$ with support in

$W(Q)$. Evaluation uses the same template and varies anchor length $n \in \{2, 3\}$ and the fraction of seed word-characters covered by islands $r \in \{0.2, 0.3\}$. The remaining generator settings are fixed. Thus $W1 = (2, 0.2)$, $W2 = (2, 0.3)$, $W3 = (3, 0.2)$, and $W4 = (3, 0.3)$.

#### D.2.1 Workload transfer

Figure 7 trains on one TPC-H workload group and tests zero-shot on the other groups under the same inference and verification setup.

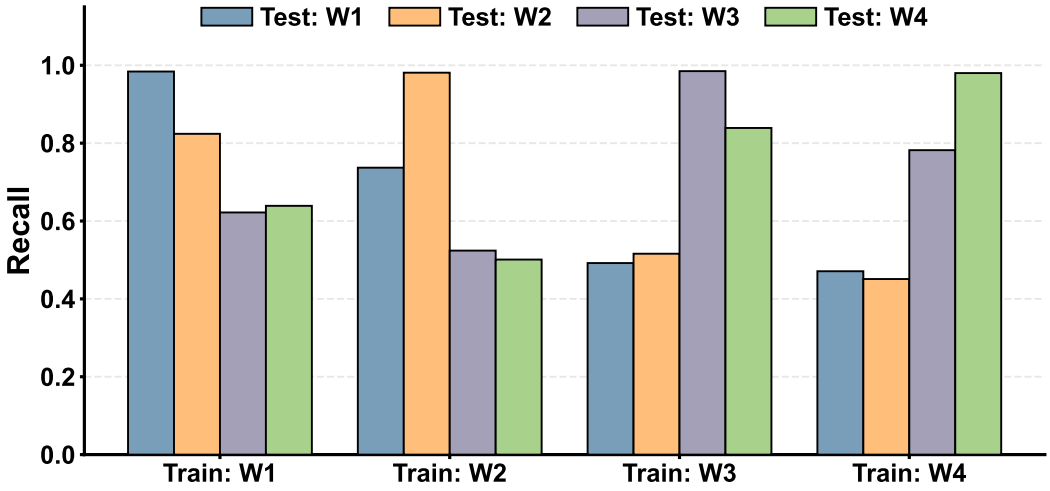



Figure 7: **Workload-transfer (OOD) on TPC-H** `lineitem.comment`**.** We train the translator on one workload group ($W_i$) and evaluate zero-shot on another ($W_j$), where $W1$–$W4$ vary anchor length $n$ and exposure budget $r$. Bars report **Recall@64** under the same inference/verification setup (S=32 samples; exact verification by the same PG regex engine).

### D.3 Baselines and metrics

SEQSCAN verifies every record; PostgreSQL trgm GIN and GiST use indexed retrieval followed by recheck. FREE retrieves candidates from extracted grams, and REI uses record-level $n$-gram bitmap filtering. BLARE splits or reorders regex matching; PNS uses anchor windows and negative factors, and PMNS-LITE adds necessary-literal pruning. All timings include candidate selection and the same native exact verifier. The non-neural column-aware ranker and its matched budget are specified in App. L.1.

Latency@$k$ is mean end-to-end query time, including inference when used, index retrieval, and verification. For unordered retrieval,

$$\text{Recall@}k = \frac{\min(k, |\widehat{S}(Q)|)}{\min(k, |W(Q)|)}.$$

Queries with no oracle matches are excluded from mean recall. This measures limit completion without requiring the scan's output order or record subset. E2E Latency@$k$ covers the full TPC-H plan. Preview Success@$\tau(k)$ is the fraction of queries returning at least $k$ verified records within $\tau$; membership tests report mean total time and rounds.

### D.4 System and timing setup

The shared server has 4×RTX A6000 GPUs, 2×Xeon E5-2620 v3 CPUs (12 cores), and 251 GB memory. PostgreSQL 16.3 uses default `shared_buffers`, `work_mem=4MB`, `jit=off`, and `max_parallel_workers_per_gather=0`. CUDA 12.x is used on A6000; the ByT5 control uses ROCm 5.x on DCU. Main measurements colocate the translator and database. Each per-query latency is the median of three measured runs after one warmup, and workload summaries aggregate these

per-query timings. The original experiments use seed 42; added controls state their matched settings locally.

# E Additional Retrieval Results

## E.1 Interactive Wikipedia-Title Query

Stack Overflow question 60224177 asks for Wikipedia titles beginning with "List of" or "Lists of" Sharma [2020]. We instantiate the "List of X in Y" subpattern on the WIKI title column:

```
List of [A-Z][A-Za-z0-9 ,:'-]+ in [A-Z][A-Za-z0-9 ,:'-]+
```

All methods use `LIMIT` 64 and return 64 exactly verified matches. The end-to-end latencies are 123.9 ms (RIVET), 658.4 ms (REI), 1,527.7 ms (BLARE), and 7,025.3 ms (SeqScan). This is a concrete query instance of the public requirement; the product workflow discussed in Sec. 2 supplies a separate example of regex-plus-limit use.

## E.2 Sensitivity to the result limit

Table 7 retains every dataset–workload cell for RIVET and REI at $k \in \{16, 32, 128\}$ under the same inference and verification budget. The $k = 64$ comparison across all methods is in Fig. 3.

Table 7: Latency (ms) of RIVET and REI across all workloads at three result limits. Values are the original per-workload measurements.

| | | $k = 16$ | | $k = 32$ | | $k = 128$ | |
|---|---|---|---|---|---|---|---|
| Dataset | Workload | RIVET | REI | RIVET | REI | RIVET | REI |
| IMDB | W1 | 32.56 | 241.86 | 34.50 | 279.43 | 37.10 | 327.75 |
| | W2 | 32.24 | 187.49 | 32.77 | 203.46 | 35.79 | 230.13 |
| | W3 | 34.27 | 169.05 | 32.45 | 178.50 | 36.77 | 217.46 |
| | W4 | 33.93 | 154.31 | 32.03 | 162.71 | 36.43 | 194.47 |
| TPC-H | W1 | 65.27 | 2217.83 | 70.40 | 2374.91 | 72.53 | 2573.61 |
| | W2 | 68.42 | 2164.07 | 69.70 | 2213.87 | 77.01 | 2416.73 |
| | W3 | 67.74 | 2051.39 | 67.93 | 2183.69 | 75.13 | 2491.30 |
| | W4 | 58.04 | 1986.43 | 61.72 | 2007.11 | 69.45 | 2281.46 |
| US-Acc | W1 | 34.18 | 1142.80 | 33.59 | 1206.48 | 37.98 | 1467.28 |
| | W2 | 33.84 | 973.67 | 32.28 | 1019.26 | 37.28 | 1276.31 |
| | W3 | 33.16 | 952.77 | 34.76 | 993.64 | 38.16 | 1263.00 |
| | W4 | 31.58 | 891.45 | 34.42 | 924.71 | 37.18 | 1047.85 |
| WIKI | W1 | 29.28 | 98.26 | 25.74 | 101.43 | 29.40 | 112.09 |
| | W2 | 28.99 | 86.43 | 27.11 | 92.76 | 30.04 | 105.73 |
| | W3 | 30.23 | 87.15 | 26.84 | 90.08 | 34.46 | 106.24 |
| | W4 | 29.93 | 89.93 | 24.39 | 94.52 | 31.28 | 108.59 |

## E.3 Interactive membership

On `imdb_name`, an agent searches for a target entity in successive `LIMIT`-16 rounds, stopping when it appears or after three rounds. RIVET takes 531 ms total versus 1,702 ms for REI and 14,324 ms for SeqScan, with mean round counts of 1.00 versus 1.11–1.35 for the baselines.

## E.4 Additional preview results

At $k = 16/32$, RIVET completes 100% of queries for all reported budgets $\tau \in \{500, 1000, 5000, 10000\}$ ms; REI completes 0.85/0.81 at 500 ms. For $k = 128$, RIVET completes 0.98 at 500 ms. Figure 5a gives the full method comparison at the default $k = 64$.

# F Backbone and Training Configurations

Tables 8 and 9 specify Exp.3. We fix each family's depth and adjust width to match trainable parameter counts, including the prediction head. The RIVET–ByT5 supervision control uses the separately listed recipe in Table 10.

Table 8: Translator backbone configurations under matched parameter budgets.

| **Backbone** | **Params (M)** | **Architecture** |
|---|---|---|
| Transformer | 30.1 | $L$=4, $d$=512, $h$=8, $d_{ff}$=2048, $p$=0.1 |
| GRU | 30.0 | layers= 2, hidden= 1072, bidir=False, p= 0.1 |
| LSTM | 30.0 | layers= 2, hidden= 928, bidir=False, p= 0.1 |
| RNN | 30.0 | layers= 2, hidden= 1712, bidir=False, p= 0.1 |
| TCN | 30.0 | layers= 4, ch= 880, k= 3, p= 0.1 |

Table 9: Training settings shared across backbones in Exp.3.

| **Setting** | **Value** |
|---|---|
| batch size | 512 |
| epochs | 400 |
| optimizer / lr | AdamW, $8 \times 10^{-4}$ |
| weight decay | 0.01 |
| grad clip | 1.0 |
| accumulation steps | 4 |
| scheduler | OneCycle (pct_start=0.15, final_div_factor=100) |
| seed | 42 |

Table 10: **Training settings for the RIVET–ByT5 supervision comparison.**

| Setting | RIVET-Base | ByT5-Small (FT) |
|---|---|---|
| Optimizer | AdamW | AdamW |
| Peak LR | $3\times10^{-4}$ | $1\times10^{-4}$ |
| Schedule | linear warmup, cosine | linear warmup, constant |
| Batch size | 256 | 64 |
| Training steps | $\sim$60k | $\sim$30k |
| Repeated-hits ratio | 0.2 | 0.2 |
| Random seed | 42 (fixed) | 42 (fixed) |
| Hardware | 1$\times$A6000 | 1$\times$DCU (ROCm) |
| Wall-clock | $\sim$2,300 s | $\sim$13,200 s |

# G Real-World OOD Workload Construction

This appendix documents how the 80 human-authored regex queries used in Sec. 5.4 and App. H were collected, cleaned, and quantitatively contrasted with the W1–W4 synthesis workloads.

## G.1 Source Corpora

We assemble 20 queries per dataset (80 in total) from two public sources:

- **StackOverflow Q&A.** Questions tagged `regex` together with one of the dataset's domain tags (e.g., `names`, `wikipedia`, `logging`, `tpch`). For each accepted answer we extract the regex literal that solves the asker's stated problem.
- **arXiv documents.** Pre-print sources from data-management and IR venues that cite RegExLib-style mini benchmarks; we include regexes that were authored to match patterns in publicly described corpora resembling our four columns.

For each query we record the regex, its category, source, the original-source URL, the original (pre-adaptation) regex string, and an adaptation note describing any change made during PostgreSQL adaptation.

### G.2 Cleaning and PostgreSQL Adaptation

The original regexes were written for engines such as PCRE/Python `re`; we adapt them to PostgreSQL's POSIX-style regex (`~`) operator while preserving match semantics: (1) lookarounds `(?=...)/(?!...)` are unsupported and are either rewritten or removed with the change recorded in `adaptation_note`; (2) word-boundary `\b` is replaced with explicit non-word-character context `[^[:alnum:]_]`; (3) anchors `^` and `$` are preserved exactly; (4) inline flags such as `(?i)` are removed and case sensitivity is matched at the operator level; (5) the `\d`/`\w` shorthands are expanded to POSIX classes `[0-9]`/`[[:alnum:]_]`. After adaptation, every query is verified to compile against the target column under PostgreSQL 16.3.

### G.3 Quantitative Distribution Shift vs. W1–W4

To make the OOD claim falsifiable, we compute five regex-AST features per query and compare the OOD distribution against the W1–W4 training distribution per dataset: (i) *character-class density* (fraction of characters that lie inside a `[...]` class); (ii) *anchor density* (count of `^`/`$` per character); (iii) *Kleene-closure count* (number of `*`/`+`/`{...}` quantifiers); (iv) *alternation depth* (maximum nesting of `|`); (v) *mean regex length*. The OOD set has substantially higher character-class density and anchor density on every dataset—e.g., on IMDB the mean character-class density is $\sim$0.55 for OOD vs. $\sim$0.10 for W1–W4—which directly explains why prefix-based key extraction is harder on OOD: an anchored character-class regex like `^[A-Z][a-z]+( [A-Z][a-z]+)*$` contains no contiguous literal of length $\geq 3$ for any prefix-extraction heuristic to exploit.

## H Real-World OOD Generalization Results

This appendix expands Sec. 5.4: aggregate three-way comparison, baseline routing statistics, AST-based routability analysis, a representative per-query table, and an oracle key-primitive analysis on the same OOD queries.

### H.1 Aggregate Three-Way Comparison

Table 11 reports Recall@64 and mean/median latency for RIVET, BLARE, and REI on the same 80 OOD queries (4 datasets $\times$ 20). All numbers use $k$=64, $t_{\max}$=32 for RIVET, the budgeted decoder used in Fig. 3. SEQSCAN reference Recall@64 is 1.000 by construction.

Table 11: **Original fixed-limit OOD benchmark.** Per-method Recall@64 and latency on 80 human-authored regex queries. The additional matched-output and residual-scan comparisons are reported separately below and in Table 2.

| Dataset | Method | R@64 | Mean ms | Median ms |
|---|---|---|---|---|
| IMDB | RIVET | 0.866 | 54 | 35 |
| | BLARE | 1.000 | 2,013 | 4 |
| | REI | 0.750 | 651 | 412 |
| US-Acc | RIVET | 0.804 | 67 | 48 |
| | BLARE | 1.000 | 5,959 | 7,208 |
| | REI | 1.000 | 1,257 | 982 |
| WIKI | RIVET | 0.622 | 48 | 34 |
| | BLARE | 1.000 | 910 | 35 |
| | REI | 0.950 | 286 | 198 |
| TPC-H | RIVET | 0.878 | 92 | 71 |
| | BLARE | 1.000 | 13,471 | 2 |
| | REI | 1.000 | 3,105 | 2,015 |

The query-level distribution of recall (Table 12) separates scan-backed exact retrieval from low-latency budgeted retrieval. BLARE's uniformly high recall is tied to its direct-scan route (App. H.3), while RIVET's OOD coverage profile aligns with AST patterns analyzed in App. H.4.

Table 12: **Query-level Recall@64 bands** over the 80 OOD queries.

| Method | High band | Middle band | AST-routed band |
|---|---|---|---|
| BLARE | 80 | 0 | 0 |
| REI | 74 | 0 | 6 |
| RIVET | 57 | 11 | 12 |

### H.2 Matched-Output and Full-Coverage Comparisons

For each query $q$, let $h(q)$ be the number of verified matches returned by RIVET's fixed-budget path. Each baseline then runs with `LIMIT` $h(q)$, using the same end-to-end timing protocol. Table 13 compares the cost of reaching this common output target. The full-coverage experiment instead keeps $k = 64$ and adds residual scanning to RIVET when needed; Table 2 reports recall alongside latency for each path.

Table 13: Matched-output OOD latency (ms), with per-query `LIMIT` $h(q)$.

| Dataset | RIVET | SeqScan | REI | BLARE |
|---|---|---|---|---|
| IMDB | 54.37 | 19,393.28 | 651.42 | 2,013.67 |
| US-Acc | 67.18 | 46,179.54 | 1,257.36 | 5,959.81 |
| WIKI | 48.26 | 6,665.47 | 286.15 | 910.63 |
| TPC-H | 92.41 | 145,176.32 | 3,105.58 | 13,471.74 |

At the shared output targets, REI is 5.9–33.6× slower and BLARE 18.9–145.8× slower than RIVET. Residual scanning reaches Recall@64 $= 1$ on all four columns. The fastest complete-coverage method differs by column: RIVET + scan on IMDB, REI on US-Acc and TPC-H, and BLARE on WIKI. An application can select its access path using estimated cost and its required coverage; the tables report the measured operating points of each method.

### H.3 Baseline Routing-Mode Statistics

Table 14 separates index-filtering behavior from scan-backed routing behavior for BLARE and REI.

Table 14: **Routing-mode counts** on the 80 OOD queries. "BLARE direct" = literal extractor returned $\emptyset$, system uses the direct-scan route. "REI mask=0" = bigram signature routes the query to full-table verification.

| Dataset | BLARE | | REI | |
|---|---|---|---|---|
| | direct | prefilter | mask=0 | mask>0 |
| IMDB | 18 | 2 | 15 | 5 |
| US-Acc | 5 | 15 | 4 | 16 |
| WIKI | 13 | 7 | 11 | 9 |
| TPC-H | 7 | 13 | 11 | 9 |
| **Total** | **43/80** (54%) | 37/80 | **41/80** (51%) | 39/80 |

The counts separate mechanism-level filtering from scan-backed routing. On more than half of the OOD queries, BLARE reaches the direct-scan path and REI reaches the full-table CTE path; these routes account for much of their recall on structurally difficult patterns. The same query subset also clarifies where prefix-guided retrieval is most effective: all three methods benefit when the regex exposes literal or $n$-gram signal that can be turned into selective access.

### H.4 AST-Based Routability Analysis

The 12-query AST-routed subset shares features computable offline, without reference to the data column:

- **(i) Pure character-class with no contiguous literal $\geq 3$ chars.** Example: `^[A-Z][a-z]+( [A-Z][a-z]+)*$` (IMDB), `[A-Z][-'A-Za-z]+,? [A-Z][-'A-Za-z]{1,19}` (IMDB).
- **(ii) Full-string anchored with non-prefix-bounded match.** Example: `^[A-Z][...] \([^)]*\)$` (WIKI), where the discriminative content sits in the parenthetical suffix.
- **(iii) Literals only in suffix or infix position.** Example: `At .+ Exit .+ - Accident` (US-Acc), where "Exit" and "Accident" are informative but never appear in any prefix of the matching record.

Table 15: **AST-routability counts** for the 12-query routed subset.

| Pattern | IMDB | US-Acc | WIKI | TPC-H |
|---|---|---|---|---|
| (i) Pure char-class | 2 | 0 | 4 | 0 |
| (ii) Full-string anchor, non-prefix | 0 | 1 | 2 | 0 |
| (iii) Literal in suffix/infix only | 1 | 2 | 0 | 0 |

All three patterns are deterministic functions of the regex AST and can be flagged offline at compile time. A deployment can use these predicates to route each query either to RIVET's prefix-guided path or to a direct-scan path.

### H.5 Per-Query Detail

Table 16 reports the query-level Recall@64 distribution per dataset, partitioned into a high band, a middle band, and an AST-routed band.

Table 16: **Per-query Recall@64 bands** on the 80 OOD queries (counts per band).

| Dataset | High band | Middle band | AST-routed band | Total |
|---|---|---|---|---|
| IMDB | 14 | 4 | 2 | 20 |
| US-Acc | 14 | 3 | 3 | 20 |
| WIKI | 11 | 3 | 6 | 20 |
| TPC-H | 18 | 1 | 1 | 20 |
| **Total** | **57** | **11** | **12** | **80** |

### H.6 Oracle Key-Primitive Analysis

To quantify the contribution of the key primitive, we compute an oracle upper bound: for each of the 80 OOD queries, we (a) collect all true matches against the column, (b) extract length-$\ell$ keys (prefix or contiguous substring) from those true matches, and (c) apply greedy set cover with budget $t_{\max}$=32 to pick the best key set.

Table 17: **Oracle match coverage** under prefix vs. substring keys, $t_{\max}$=32, averaged over the 80 OOD queries.

| Key primitive | Avg. match coverage |
|---|---|
| Prefix (current RIVET) | 38.8% |
| Substring | **60.6%** (1.56×) |

The $1.56\times$ oracle comparison in Table 17 characterizes the additional coverage available from substring-style keys at the same budget. Prefix keys are the evaluated instantiation because they (i) reuse standard B-tree indexes via `LIKE 'p%'`, (ii) deliver bounded per-query latency, and (iii) yield

disjoint candidate buckets. Substring keys define a complementary design point with a different index primitive (e.g., suffix-array or generalized n-gram), and the oracle result shows that the Key+Verify framework can support such primitives.

# I Cost Decomposition and Break-Even Analysis

This appendix expands Sec. 5.3 with full one-time cost numbers, online cost decomposition, in-distribution and OOD break-even analyses, hyperparameter sensitivity, and the scan-backed routing analysis.

## I.1 Offline (One-Time) Cost

Table 18: **Per-column offline cost**, measured on a single A6000 host.

| Method | Offline cost | Storage |
|---|---|---|
| RIVET (training) | $\sim$2,300 s ($\sim$39 min) | 58–60 MB FP16 |
| RIVET shared B-tree | reused across methods | 78–884 MB |
| trgm GIN (BLARE prefilter) | 85–1,338 s build | 106–800 MB |
| REI signature rebuild | 47 s (WIKI) – 232 s (TPC-H) | 260 MB–3.3 GB |

The shared `text_pattern_ops` B-tree used by RIVET for prefix lookup is also queryable directly without a model and is treated as a database-side reusable structure; we report it separately to avoid double-counting. trgm GIN sizes apply to BLARE's prefilter; REI maintains its own signature table whose sizes scale with column cardinality.

## I.2 Online (Per-Query) Cost Decomposition

RIVET's per-query latency decomposes into model inference, prefix lookup over the shared B-tree, and exact regex verification. For the batched, fixed-budget execution measured here ($t_{\max}$=32), median per-query inference is $\leq$ 10 ms on the A6000, while verify cost dominates and scales with the concentration of true matches in the prefix-bucketed candidate set. Table 19 summarizes per-component median latency on the OOD workload.

Table 19: **RIVET online latency decomposition** on the OOD workload (medians, ms).

| Stage | IMDB | US-Acc | WIKI | TPC-H |
|---|---|---|---|---|
| Model inference | 6 | 8 | 5 | 9 |
| Prefix lookup | 12 | 15 | 11 | 22 |
| Verify | 17 | 25 | 18 | 40 |
| Total (median) | **35** | **48** | **34** | **71** |

## I.3 Break-Even Analysis (In-Distribution and OOD)

Because RIVET trades a one-time training cost for per-query savings, we report two break-even calculations: one using the in-distribution per-query latencies of Fig. 3, and one using the original fixed-budget OOD timings. The amortization point is computed against the strongest baseline (REI) as queries_to_amortize $= T_{\text{train}}/(\ell_{\text{REI}} - \ell_{\text{RIVET}})$ with $T_{\text{train}}$=2,300 s.

The OOD break-even is consistently *lower* than the in-distribution break-even: REI's bigram mask routes 51% of OOD queries through full-table CTE verification, whereas RIVET keeps the prefix-guided budget fixed across regex shapes.

**Residual-scan cost.** Table 2 reports the key path plus residual scan, including its total latency and final Recall@64 of 1.000 on every column.

Table 20: **Break-even queries vs. REI** under in-distribution (Fig. 3) and the original fixed-budget OOD latencies.

| Dataset | In-distribution | OOD |
|---|---|---|
| IMDB | ∼12,200 (2.4 d @ 5K/d) | ∼3,850 (0.77 d) |
| US-Acc | ∼2,144 (0.43 d) | ∼1,950 (0.39 d) |
| WIKI | ∼31,500 (6.3 d) | ∼9,650 (1.93 d) |
| TPC-H | ∼1,032 (0.21 d) | ∼750 (0.15 d) |

Table 21: Deployment measurements. CPU timing covers translator inference only; payload includes request and response.

| Measurement | Result |
|---|---|
| CPU translator | Median 153.6 ms; p90 181.9 ms |
| RPC payload | Median 296 B; maximum 607 B |
| WIKI-W4 scan/RIVET crossover | Between 5K and 10K rows |

### I.4 CPU Inference, RPC Payload, and Small Tables

A separated translator adds approximately one network round trip. At 1 Gbps, the maximum observed 607 B payload has under 0.005 ms serialization time, so network RTT determines the transfer contribution. The CPU measurement isolates model inference; index retrieval and verification complete the query.

The WIKI-W4 experiment varies table size from 5K to 2M rows, holding the regex workload, model, key budget, and execution paths fixed. SeqScan is faster at 5K rows and RIVET at 10K. This measured interval identifies where the saved scan work outweighs inference and index overhead for that workload.

### I.5 Hyperparameter Sensitivity ($t_{\max}$ and $\ell$)

**Key budget $t_{\max}$.** Table 22 shows Recall@64 across $t_{\max} \in \{4, 16, 32, 64\}$. Quality plateaus for $t_{\max} \geq 16$ on every dataset, with within-$\pm$2pp variation; the default $t_{\max}$=32 sits comfortably inside the plateau. End-to-end latency varies by less than 10% across this range thanks to dictionary filtering and sample parallelism.

Table 22: **Recall@64 vs. $t_{\max}$.**

| $t_{\max}$ | IMDB | WIKI | TPC-H | US-Acc |
|---|---|---|---|---|
| 4 | 0.910 | 0.934 | 0.851 | 0.906 |
| 16 | 0.991 | 0.976 | 0.982 | 0.985 |
| 32 (default) | **0.986** | **0.985** | **0.985** | **0.985** |
| 64 | 0.997 | 0.986 | 0.991 | 0.991 |

**Prefix length $\ell$.** The optimal $\ell$ is a per-column property: it should match the column's typical string-length distribution. We use $\ell \in \{6, 8, 10\}$ across the four datasets, set once at training time per column, and observe Recall@64 within $\pm$1.5pp around the chosen value for $\ell \pm 2$. No per-query tuning is required.

## J Data Efficiency and Maintenance

### J.1 Cold-start data efficiency

We train on 100, 1K, and 10K available WIKI records with architecture, training budget, and decoder fixed, and compare against Full on the same held-out WIKI workload. Table 23 gives the original values; Full denotes the full-data reference rather than a numeric sample count.

Table 23: WIKI cold start. Full denotes the full-data reference setting.

| Available records | 100 | 1K | 10K | Full |
|---|---|---|---|---|
| Recall@64 | .180 | .345 | .714 | .985 |
| Relative to Full | 18.3% | 35.0% | 72.5% | 100% |

A deployment can activate RIVET once held-out recall reaches its target, using REI or SeqScan during initial training.

### J.2 Maintenance under drift

Figure 5b reports the three policies on the time-ordered `lineitem.comment` stream: NoUpd retains Recall@64 of 0.87–0.89; FastUpd takes 119 s per partition and reaches 0.90–0.94; LongUpd takes 1,170 s and reaches 0.98–0.99. The evaluated trigger uses the SeqScan-oracle subset in a 500-query sliding window and fires FastUpd below recall 0.92. LongUpd follows sustained drift after two consecutive FastUpds. Offline index rebuild costs are reported in App. I.1.

## K Additional LLM and Pretrained Backbone Comparisons

This appendix expands the LLM and pretraining controls in Exp.10 of Sec. 5.4.

### K.1 Zero-Shot LLM Prefix Generation

We prompt four off-the-shelf LLMs (DeepSeek-V3.2 671B, Qwen3.5-35B-A3B, Qwen3.5-9B, Qwen3.5-4B) to generate length-$\ell$ prefix keys for the same regex queries on which RIVET is evaluated, taking the better of two prompt strategies (in-context examples vs. chain-of-thought). The metric we report is the *dictionary hit rate*: the fraction of generated keys that exist as a length-$\ell$ prefix in the target column. End-to-end latency is the response time of a hosted inference API.

Table 24: **LLM prefix-generation hit rate vs. RIVET.** Each entry is the fraction of generated prefix keys that exist in the target column's prefix dictionary; latency is end-to-end per-query inference time.

| Method | Params | IMDB | TPC-H | WIKI | US-Acc | **Avg.** | Latency |
|---|---|---|---|---|---|---|---|
| DeepSeek-V3.2 | 671B | 42.1% | 22.0% | 40.2% | 0.4% | 26.2% | 7.6 s |
| Qwen3.5-35B-A3B | 35B | 39.3% | 19.6% | 47.6% | 0.1% | 26.7% | 6.7 s |
| Qwen3.5-9B | 9B | 38.8% | 33.3% | 43.9% | 0.6% | 29.2% | 20.5 s |
| Qwen3.5-4B | 4B | 39.5% | 47.7% | 41.9% | 2.0% | 32.8% | 41.6 s |
| **RIVET** | **30M** | **98.2%** | **99.1%** | **97.4%** | **99.3%** | **98.5%** | **$\sim$40 ms** |

On US-Acc, all four LLMs attain $\leq 2\%$ dictionary-hit rate because the column vocabulary consists of incident-report templates from a domain-specific feed. On the strongest LLM setting in this comparison (Qwen3.5-4B on TPC-H, $47.7\%$), fewer than half of the generated keys correspond to indexable prefixes. This supports the central design point: the prefix distribution that drives selectivity is column-specific and is learned from the target column.

### K.2 Fine-Tuned ByT5-Small

To isolate whether *character-level pretraining* (rather than just zero-shot transfer) helps, we fine-tune ByT5-Small ($\sim$300 M params, character-level) on the IMDB workload using the same supervision and the same number of training steps as RIVET-Base, with the training settings in Table 10. ByT5's prefix-key accuracy plateaus at $53.6\%$ vs. RIVET-Base's $98.9\%$ under matched supervision; ByT5's per-query inference latency is approximately $15\times$ slower under matched decoding. The continued decrease in training loss after the accuracy plateau is consistent with the difference between ByT5's denoising pretraining objective and the target column's prefix distribution.

**Hardware note.** ByT5 fine-tuning was conducted on a Hygon DCU host with ROCm runtime; all other RIVET / LLM / baseline experiments use a CUDA / A6000 host. We report this hardware

distinction because per-query accuracy and dictionary-hit-rate metrics are hardware-independent, and the conclusions in Sec. 5.4 rest on those metrics.

# L Component Ablation Details

This appendix expands the component evidence in Exp.2 of Sec. 5.2. We isolate each piece of the online pipeline by replacing it with a controlled variant while keeping the rest fixed.

Table 25: **Online-pipeline component ablation.** Recall@64 across four datasets at $k$=64, $t_{\max}$=32.

| Variant | IMDB | WIKI | TPC-H | US-Acc |
|---|---|---|---|---|
| Random keys (column dict.) | 0.005 | 0.002 | 0.116 | 0.009 |
| Greedy decoding (single key) | 0.489 | 0.624 | 0.448 | 0.688 |
| Beam search only | 0.945 | 0.947 | 0.952 | 0.961 |
| **Full pipeline** (beam + temperature) | **0.986** | **0.985** | **0.985** | **0.985** |

The random-keys baseline samples $t_{\max}$=32 keys uniformly from the column's prefix dictionary $\mathcal{K}(D)$, measuring the value of learned key selection. Its Recall@64 range (0.002–0.116) shows the sparsity of useful prefix buckets under uniform probing; the learned regex-to-prefix mapping provides a $\sim$197$\times$ Recall@64 lift on average. The greedy variant uses a single best-prefix decode, which captures less multi-bucket coverage than set-valued prefix retrieval (a single regex usually has multiple equally-valid matching prefixes); beam search recovers most of this coverage by exposing diverse high-likelihood candidates, and adding temperature sampling fills the remaining $\sim$3% tail by exploring lower-likelihood but still in-dictionary prefixes.

## L.1 Column Specialization and Supervision Controls

The cross-column and pooled controls hold architecture, decoder, key budget, and target test queries fixed. Best cross-column transfer selects the best specialist trained on another column. Compute-matched pooling uses the same total training budget as a specialist; exposure-matched pooling uses four times that budget so each column receives the specialist's exposure. Table 1 reports the per-column results: the average recall deficits are 29.0, 7.2, and 2.3 percentage points for direct transfer, compute-matched pooling, and exposure-matched pooling, respectively.

The IMDB supervision control keeps column data, architecture, training volume, and key budget fixed and shuffles only the regex–prefix pairing. Recall@64 falls by 39.1 percentage points, linking retrieval quality to the paired supervision. The non-neural column-aware ranker uses bucket cardinality, sampled regex hit rate, AST compatibility, and estimated yield at the same key budget, obtaining 0.338 Recall@64 against 0.985 for RIVET. Together these controls identify the contribution of query-conditioned, column-specific keys.

# M Scope and Operating Regime

This appendix summarizes the measured operating regimes.

**Per-column training.** RIVET learns from records in the target column. Cross-column and pooled controls appear in App. L.1, and cold-start data efficiency in App. J.1. Training amortization depends on the workload: at 5K queries/day the original fixed-budget OOD comparison gives 0.15–1.93 days, while in-distribution WIKI requires 6.3 days (App. I.3).

**Deployment.** Main latency measurements use colocated inference and PostgreSQL. App. I.4 reports CPU-only inference, RPC payload, and the WIKI-W4 crossover between 5K and 10K rows. Main table sizes span 2.0M–24.0M rows.

**Coverage.** On 80 human-authored OOD regexes, the fixed-budget path reaches Recall@64 of 0.622–0.878; residual scanning reaches 1.000 (Table 2). These paths serve preview and filled-limit requirements, respectively.

**Analysis.** The inclusion and hit-probability formulas concern independent stochastic proposals before truncation. Theorem C.3 gives an order-free candidate-count bound and a conditional expected cost under uniform random verification order. The empirical in-distribution validation is reported in App. C.8; OOD behavior is measured in App. H.

## NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes].

   Justification: The abstract and introduction state the paper's main claims for regex-to-indexable-key neural translation for LIMIT-k retrieval, and the theory, experiments, and appendix delimit the corresponding contribution scope.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes].

   Justification: Section 6 summarizes coverage, data, maintenance, and deployment conditions; the corresponding experiments and appendix provide quantitative details.

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best

judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [Yes].

   Justification: The paper states formal results with their assumptions and provides proof sketches or full proofs in the main text and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include theoretical results.
   - All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
   - All assumptions should be clearly stated or referenced in the statement of any theorems.
   - The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
   - Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
   - Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes].

   Justification: The main text and appendix disclose the models or datasets, protocols, metrics, baselines, and experimental settings needed to reproduce or verify the main claims.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).

(d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [Yes].

   Justification: A reference implementation of the translator, the in-distribution workload generator, the end-to-end evaluation pipeline, per-experiment training entrypoints, and the baseline drivers used for BLARE and REI are provided in the supplementary archive accompanying the submission. Index, hyperparameter, decoding, hardware, PostgreSQL, and protocol settings are documented in App. D–D.4.

   Guidelines:

   - The answer [N/A] means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
   - At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
   - Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

   Answer: [Yes].

   Justification: The experiments specify the relevant task construction, splits or sampling protocol, model settings, decoding/training parameters, baselines, and evaluation metrics in the main text and appendix.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [No].

Justification: The evaluation uses one training seed and median-of-three per-query timings (App. D.4). The paper reports workload and parameter-sensitivity results, but does not report confidence intervals or significance tests for the main comparisons.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes].

   Justification: The appendix specifies the shared server CPU/GPU/memory, database/runtime configuration, per-experiment hardware and wall-clock, median-over-runs latency protocol, offline build/storage costs, and training amortization reference.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes].

   Justification: The work uses public or research datasets/models where applicable, and the authors have checked conformance with the NeurIPS Code of Ethics.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.

- The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [N/A].

    Justification: The work is primarily a foundational or diagnostic study of regex-to-indexable-key neural translation for LIMIT-k retrieval and does not introduce a direct user-facing deployment artifact or application-specific system with clear societal-impact claims.

    Guidelines:

    - The answer [N/A] means that there is no societal impact of the work performed.
    - If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A].

    Justification: The paper does not release a new high-risk model, scraped dataset, or deployment artifact requiring controlled-access safeguards.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

Answer: [No].

Justification: The manuscript identifies datasets, baselines, database/index assets, pre-trained/hosted models, and URLs for several sources, but it does not list licenses and terms of use for each existing asset. Asset sources and versions are listed in App. D.1.

Guidelines:

- The answer [N/A] means that the paper does not use existing assets.
- The authors should cite the original paper that produced the code package or dataset.
- The authors should state which version of the asset is used and, if possible, include a URL.
- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

Answer: [Yes].

Justification: The reference implementation provided in the supplementary archive is documented in the manuscript and appendix: model architecture and training protocol (App. B.3); workload-generation template (App. D.2); decoding parameters, hyperparameters, and seeds (App. F); environment, hardware, and PostgreSQL settings (App. D.4, App. D.4). No new public dataset, benchmark, or pretrained model is released as a stand-alone artifact.

Guidelines:

- The answer [N/A] means that the paper does not release new assets.
- Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
- The paper should discuss whether and how consent was obtained from people whose asset is used.
- At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

Answer: [N/A].

Justification: The work does not involve crowdsourcing or human-subject experiments.

Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A].

    Justification: The work does not involve human-subject experiments, so IRB approval is not applicable.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes].

    Justification: LLMs or LLM-derived systems are an important component of the method, experiments, or objects of study, and the manuscript specifies the relevant model families and protocols.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.