Bridging the Gap: Enabling Natural Language Queries for NoSQL Databases through Text-to-NoSQL Translation

Jinwei Lu[1], Jiawei Lu[1], Chen Zhang[1], Zhiqian Qin[1],
Yuanfeng Song[2], Haodi Zhang[3], Raymond Chi-Wing Wong[4]
[1]The Hong Kong Polytechnic University, Hong Kong, China
[2]WeBank Co., Ltd, Shenzhen, China [3]Shenzhen University, Shenzhen, China
[4]The Hong Kong University of Science and Technology, Hong Kong, China

***Abstract*—NoSQL databases are core data infrastructure, yet natural-language access to them remains underdeveloped: correct query generation must recover how a non-relational data model represents entities, nested paths, arrays, missing fields, and dynamic keys. This paper studies Text-to-NoSQL, translating natural-language requests into executable NoSQL queries, instantiated with MongoDB aggregation pipelines over schema-less document stores. We present TEND (short for Text-to-NoSQL Dataset), an execution-verified benchmark with 1,210 MongoDB-native tasks across 11 databases. To our knowledge, TEND is the first Text-to-NoSQL benchmark whose database worlds are MongoDB-native by design: experts manually define collection boundaries, nested arrays, optional and sparse paths, polymorphic shapes, and dynamic-key conventions; these worlds are populated with real data and verified through frozen MongoDB execution, so TEND evaluates schema-less document reasoning rather than SQL-to-MQL transfer. We further introduce SAG, a Schema-as-Data Grounding solver that induces path and value grounding from stored-document evidence before bounded MQL generation, execution-grounded repair, and result-consistency selection. Evaluation uses bounded column-tolerant execution accuracy (EXC) as the headline metric, complemented by a graded result-set F_1 and a mutually exclusive execution-outcome decomposition. Experiments show that LLMs with strong NL2SQL performance degrade substantially on TEND, validating Text-to-NoSQL as a distinct schema-less document reasoning problem.**

***Index Terms*—NoSQL Database, Text-to-NoSQL, MongoDB Aggregation, Benchmark, Execution Evaluation**

I. Introduction

NoSQL databases have become a central part of modern data management because they support flexible records, horizontal scaling, and heterogeneous application state without forcing every workload into fixed relational tables, and the database community has studied them from multiple angles, including data modeling, scalability, storage management, and distributed execution [2]–[8]. NoSQL is an umbrella term: document, graph, key-value, and wide-column systems relax relational assumptions in different ways and expose different query languages. This paper focuses on the document-store branch through MongoDB, a practically consequential representative whose aggregation framework makes schema-less document reasoning central to query generation [9].

Natural-language database interfaces have made substantial progress for relational databases, where Text-to-SQL can rely on explicit table-column catalogs; the NoSQL setting remains much less systematized, and the gap is deeper than query-language syntax. For MongoDB, correct aggregation pipelines often depend on optional paths, arrays that must be unwound before grouping, keys that are themselves data values, and aliases introduced by earlier pipeline stages. Many of these structures are witnessed by stored documents rather than guaranteed by a fixed catalog, so direct transfer from Text-to-SQL or SQL-to-MQL syntax conversion misses the central database challenge.

We therefore study **Text-to-NoSQL**: translating a natural-language query (NLQ) into an executable query over a concrete NoSQL database instance (Figure 1). The term names the broader problem family across non-relational data models and target languages; this paper instantiates it with MongoDB aggregation pipelines over schema-less document stores, where the technical problem is executable query synthesis under schema uncertainty, nested structure, dynamic-key evidence, and instance-witnessed semantics rather than MongoDB-like surface syntax.

To support systematic advancement in this field, we release the **TEND** dataset, which addresses the benchmark gap for MongoDB-native Text-to-NoSQL. Existing resources that touch MongoDB commonly inherit relational regularity by translating schemas or gold SQL into MQL targets; such construction is useful for syntax transfer but does not make document modeling itself the object of evaluation. TEND instead uses BIRD mini-dev [10] only as source evidence for realistic domains, values, and workload pressure. Experts then design a new MongoDB DataWorld for each selected domain, deciding collection boundaries, embedded records, array nesting, dynamic-key maps, optional paths, sparse fields, polymorphic shapes, and value witnesses by hand. Each task is paired with an expert-refined MongoDB aggregation pipeline, a canonical English request, and frozen witness data for execution verification; Section II details the construction protocol

A demonstration version of this work, Querycraft, has been accepted to the VLDB 2026 Demo Track [1].

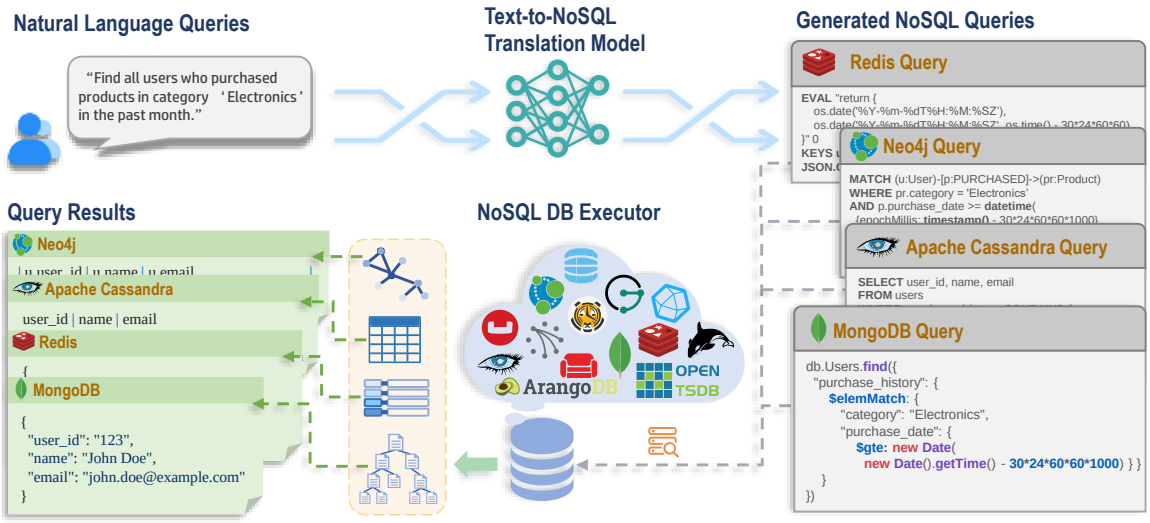

Fig. 1. The Text-to-NoSQL problem family: natural-language requests are translated into executable queries across NoSQL data models and query languages (Redis, Neo4j, Cassandra, and MongoDB illustrated); TEND and SAG instantiate the document-store branch with MongoDB aggregation pipelines.

and statistics.

The schema-as-data challenge also shapes the solver. We develop **SAG**, a Schema-as-Data Grounding solver for Text-to-NoSQL. SAG treats stored documents as the evidence from which the operative schema is induced. Before generation, it performs bounded witness sampling, builds a `GroundingIndex`, renders a closed path card with dynamic-key abstractions, anchors literals through value witnesses, and then uses bounded decode/repair with execution feedback and result-space consistency to produce a MongoDB aggregation pipeline. The design is intentionally not an open-ended exploratory agent: database grounding is performed deterministically before generation, while the loop is reserved for repairing concrete violations exposed by grounding and execution.

Finally, Text-to-NoSQL needs evaluation that reflects both executable database behavior and diagnosable query-generation failures. We use bounded column-tolerant execution accuracy (**EXC**) as the headline execution metric because MongoDB aggregation outputs may contain benign helper fields while still needing strict row, value, nested-object, and order semantics. We complement it with a graded result-set F_1 (**EXF$_1$**) that measures how close a wrong answer is to the gold result, and with a mutually exclusive outcome decomposition that attributes every failure to a single cause. Extensive experiments show that LLMs that perform strongly on NL2SQL benchmarks degrade sharply on TEND, indicating that TEND is not a surface-language variant of Text-to-SQL but a benchmark for schema-less document reasoning (Section IV).

The primary contributions of this paper are summarized as follows:

- We formalize MongoDB-native Text-to-NoSQL as executable aggregation-pipeline synthesis over schema-less document stores, keeping Text-to-NoSQL as the broader problem family.
- We construct TEND[1], to our knowledge the first MongoDB-native, expert-guided, and execution-verified Text-to-NoSQL benchmark whose database worlds are manually designed for schema-less document reasoning. TEND contains 1,210 tasks across 11 databases and makes nesting, dynamic keys, sparse and optional paths, polymorphic shapes, and witness values first-class benchmark variables.
- We design SAG, a schema-as-data grounding solver that converts stored-document evidence into a path lattice, value witnesses, alignment gates, execution-grounded repair signals, and result-consistency clusters before emitting MQL.
- We provide an execution-centered evaluation protocol with EXC as the headline metric, a graded result-set F_1 (EXF$_1$), a mutually exclusive execution-outcome decomposition, and paired significance testing, with every failure retained in the denominator.
- We set up controlled comparisons against channel-pure direct, sampled-document, relational SQL-pivot, and bounded ReAct baselines, all sharing the same backbone model and evaluator, so differences are attributable to the information channel and mechanism.

II. The TEND Benchmark

TEND is a human-annotated, execution-verified benchmark for MongoDB-native Text-to-NoSQL over schema-less document stores. Its construction is design-first: database experts author each MongoDB DataWorld and its tasks from the domain's analytic needs (Section II-B), and the BIRD mini-dev databases [10] serve only as a source of real values, domain semantics, and analytic workload pressure, never as a SQL-to-MQL translation target, schema blueprint, or execution oracle.

[1]The dataset and source code are available at https://anonymous.4open.science/r/Text-to-NoSQL-08F4/

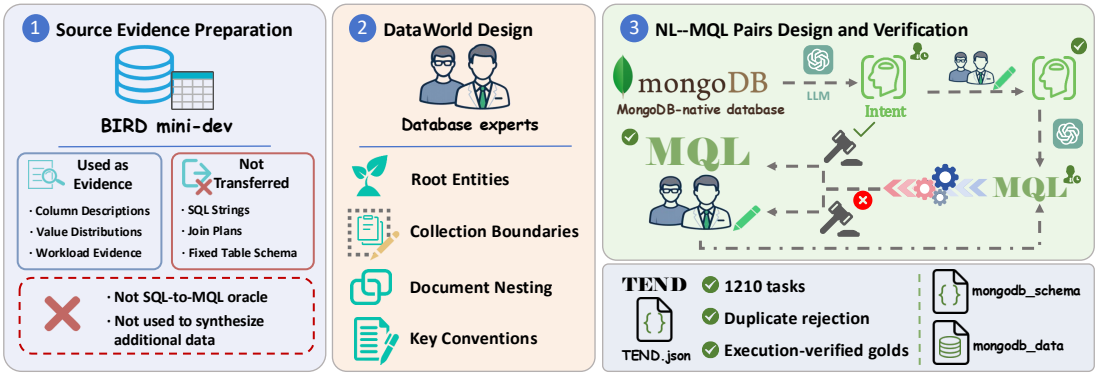

Fig. 2. Human annotation pipeline for TEND: seven database experts define one MongoDB-native construction specification per database, LLMs draft MQL programs at scale, and experts refine and execution-verify all 1,210 NL–MQL tasks.

A. Core Design of TEND

TEND is not a relational benchmark wrapped in MongoDB syntax. Its core objective is to evaluate whether a system can synthesize executable aggregation pipelines over document data whose operative schema is partly encoded in the stored instances themselves. To the best of our knowledge, TEND is the first Text-to-NoSQL benchmark whose databases are MongoDB-native DataWorlds rather than table-to-collection migrations, SQL-equivalent query targets, or generic multi-model translations.

Schema-less structure as the target of reasoning. In TEND, *schema-less* does not mean structure-free: the database provides no global catalog that fully determines every document shape, and TEND turns this flexibility into query semantics. A request may depend on a map whose keys are domain values, an embedded record carrying local evidence, an array that must be unwound before aggregation, or a field whose absence means something different from null, so the operative schema must be inferred from stored-document witnesses.

MongoDB-native DataWorlds. Native means the database design is meaningful as a document store, not merely that data are loaded into MongoDB. Experts decide collection boundaries, embedded entities, nesting depth, dynamic-key conventions, optional paths, sparse fields, polymorphic shapes, and witness structures according to domain analytic needs; BIRD rows supply realistic values, but all structural decisions are manual, making these structures benchmark variables rather than conversion artifacts.

MQL-centered difficulty. Every gold program is authored and verified as a MongoDB aggregation pipeline, not as a syntactic rewrite of a SQL query. Many tasks first build an analytic view of the document world and only then compute the requested answer, with pipelines reaching 13 stages (Section II-C). Intermediate fields, reshaped arrays, dynamic-key expansions, and derived group keys are not incidental implementation details; they are the semantic path from the natural-language request to the final result. TEND therefore tests document-level reasoning and aggregation planning, not field selection over a known schema.

Native stressors and exposed failures. TEND concentrates seven query-bearing document structures, each paired with the failure pattern it exposes in generated MQL: *dynamic-key maps* (domain values stored as object keys) expose solvers that treat keys as fixed columns and lose key–value identity; *nested arrays* expose unwinding at the wrong grain, which duplicates or drops evidence; *optional and sparse paths* expose solvers that assume uniform fields and filter out meaningful absence; *polymorphic shapes* expose commitment to a single document variant while valid alternates are ignored; *embedded records* expose invented joins that flatten away co-located context; *value witnesses* expose literals misbound to the wrong path or stored casing; and *result contracts* expose leaked helper fields, group keys, and sort artifacts. Figure 3 works one such stressor end to end.

B. Annotation Pipeline

Seven database-trained annotators proficient in MQL carry out the annotation. The pipeline (Figure 2) is design-first, authoring every MongoDB world before any source data is consulted for population, in five steps:

1) Expert DataWorld design. We author a separate MongoDB construction specification for every database from its domain analytic needs, deciding root entities, collection boundaries, embedded relationships, nesting, dynamic-key conventions, optional paths, sparse fields, and polymorphic shapes by hand. No fixed table-to-collection rule is applied, and no structural decision is inherited from the relational layout.

2) Materialization from source evidence. The expert-designed worlds are then populated with real values drawn from the 11 BIRD mini-dev databases, whose contents and workload evidence ground each domain; no data synthesis is needed, and BIRD SQL is treated only as evidence of analytic intent, never as MQL templates or execution oracles.

3) LLM-assisted NL–MQL drafting and human refinement. LLMs then mine candidate analytic intents from the source evidence and draft initial MQL programs at scale; no candidate is accepted as gold. We manually refine the intent, revise the pipeline against witness execution, and write a canonical English request aligned to the DataWorld.

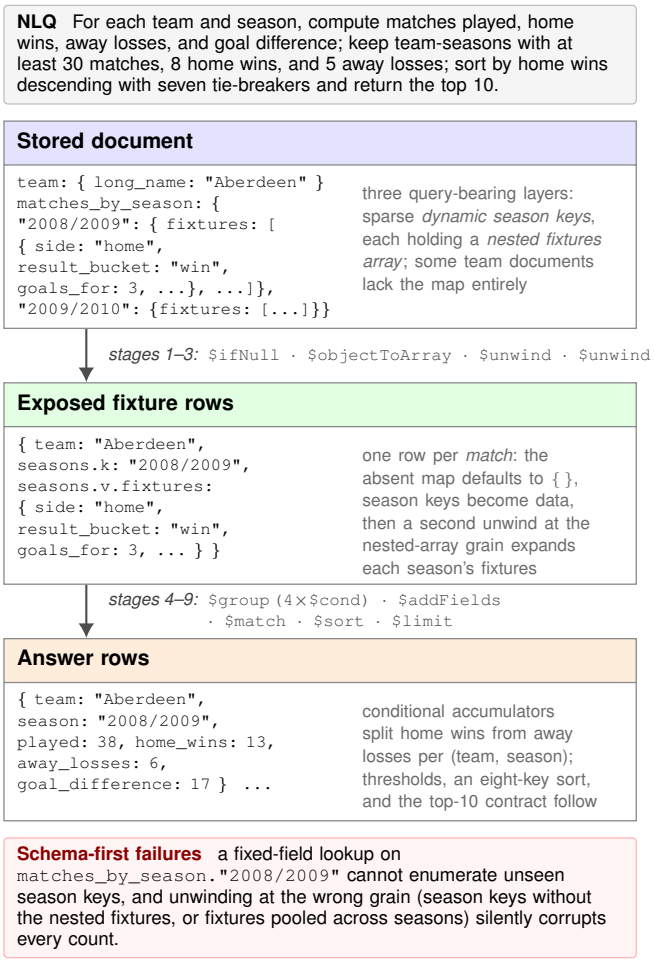

Fig. 3. A release TEND task (`european_football_2`, nine-stage gold pipeline). The requested grouping dimension exists only as sparse dynamic object keys whose values hold nested fixture arrays, so the pipeline must expose structure as data at two different grains before conditional aggregation, thresholding, and an ordered top-10 contract.

4) Execution-based verification. Every gold MQL is executed on frozen MongoDB witness data and its result inspected; the query or wording is revised and re-checked until intent, program, and execution result are mutually consistent.

5) Benchmark audit, repair, and release quality control. After construction, multiple full-benchmark audit-and-repair passes (deterministic checks plus LLM-assisted review under expert adjudication) targeted NL–MQL alignment defects, e.g., answers whose stored form or ordering is not decidable from the question, re-verifying every repaired task by execution. Deterministic validation then enforces the public record format, parses every pipeline, rejects duplicate MQL signatures, controls skeleton-family concentration, and verifies exact execution on the witness data; the release contains 1,210 validated tasks, each with one gold MQL program and one canonical English request.

C. Dataset Statistics and Characteristics

Table I summarizes TEND at release scale: 1,210 execution-distinct tasks balanced over 11 databases (110 each), each with one canonical English request, and 269,177 witness documents across 32 collections; every gold program is an aggregation pipeline with a unique MQL signature that passes exact execution on the frozen witness data. The median pipeline has seven stages and twelve distinct operators, requiring multi-stage document transformation rather than shallow filtering, and MongoDB-native stress is pervasive: 90.6% of records use dynamic-key operators, 96.7% array operators, and 96.2% nested dotted paths.

TABLE I
RELEASE-SCALE STATISTICS OF THE TEND BENCHMARK.

SCALE AND VALIDATION			
Databases	11	NL–MQL tasks	1,210
Tasks per database	110	Collections	32
Witness documents	269,177	Exact execution	1,210 / 1,210
GOLD-PROGRAM COMPLEXITY AND DIVERSITY			
Stages (med. / max)	7 / 13	Operators (med.)	12
Unique MQL signatures	1,210	Skeleton families	1,098
Largest skeleton fam.	6 (0.5%)	Skeleton entropy	0.99
NATIVE-OPERATOR USAGE (RECORDS)			
Array operators	1,170	96.7%	
Nested dotted paths	1,164	96.2%	
Dynamic-key operators	1,096	90.6%	
Grouping accumulators	652	53.9%	
CLAIM-AXIS SLICES (RECORDS)			
Schema-flex category			
Dynamic-key map	1,013	83.7%	
Nested event stream	130	10.7%	
Polymorphic shape	36	3.0%	
Missing-vs-present	18	1.5%	
Attribute bag	13	1.1%	
Anti-SQL-transfer level			
Strong	1,138	94.0%	
Medium	71	5.9%	
Weak	1	0.1%	

Program diversity is controlled at the template grain as well: a *skeleton family* groups gold programs that become identical after abstracting field names and literals; the largest family holds six of the 1,210 records and the normalized skeleton entropy is 0.99 (Table I), so TEND cannot be solved by memorizing a small set of pipeline templates.

Claim axes. Every record carries two designed labels that the evaluator uses for sliced reporting, with release distributions in Table I. The *schema-flex category* names the document-native structure the task exercises: dynamic-key maps (the dominant category), nested event streams, polymorphic shapes, missing-versus-present semantics, and attribute bags. The *anti-SQL-transfer level* grades, from gold-program operator evidence, how strongly the task resists relational transfer: a record is *strong* when its pipeline combines a strongly document-native operator (e.g., `$objectToArray` or `$switch`) with further native structure, *medium* when it uses native expressions without such combination, and *weak* when it reduces to relational-style flatten-and-group. These labels are evaluation metadata only and are never visible to any solver or baseline.

Table II summarizes these distinctions: **MongoDB-native**

TABLE II
COMPARISON OF TEND WITH EXISTING BENCHMARKS.

Dataset	Target	Scale	Native DW[a]	Schema Flex.[b]	Nested Fields	Program Diversity Control	Construction Leader[c]
WikiSQL [11]	SQL	80k+ NLQs	✗	✗	✗	✗	Program
Spider [12]	SQL	10,181 NLQs	✗	✗	✗	SQL complexity	Human
BIRD [10]	SQL	12,751 NLQs	✗	✗	✗	SQL complexity	Human
KaggleDBQA [13]	SQL	272 NLQs	✗	✗	✗	✗	Human
Spider 2.0 [14]	SQL	632 workflows	✗	✗	✗	Workflow complexity	Human
DocSpider [15]	MQL	4,663 NLQs	✗	✗	✗	Inherited from Spider	LLM+Program
SM3-Text-to-Query [16]	SQL/MQL Cypher/SPARQL	10K NLQs 40K pairs	✗	Limited	Partial	Template augmented	Human+Program
TEND	**MQL**	**1,210 NLQs**	✓	✓	✓	**1,210 unique MQL signatures**	**Human+LLM**

[a] **Native DW:** the database worlds are MongoDB-native DataWorlds authored for document semantics, rather than relational mirrors produced by one-collection-per-table migration.
[b] **Schema Flex.:** query-bearing schema flexibility: dynamic keys, optional fields, and polymorphic shapes participate in gold programs; "Limited" denotes template-level variation only.
[c] **Construction Leader:** Human: expert-authored; Program: rule-based generation; Human+Program: templates with programmatic augmentation; LLM+Program: LLM-translated gold gated by execution checks; Human+LLM: human-led with AI assistance and expert validation.

DataWorld design (collections, nesting, and query semantics authored for MongoDB rather than inherited from relational catalogs or one-collection-per-table migrations), **query-bearing schema flexibility** (dynamic keys, optional fields, arrays, and nested paths participate in gold MQL programs rather than appearing as passive irregularities), and **program-diversity control** (1,210 unique MQL signatures, ruling out template-like generation). The construction is human-led with LLM assistance, followed by expert validation and frozen MongoDB execution.

D. Evaluation Metrics

TEND uses an execution-grounded evaluator rather than relying only on textual similarity to the gold program. For each benchmark record, let q_p denote the predicted MongoDB program, q_g the gold MQL program, and D the frozen MongoDB witness database. The evaluator parses q_p, checks it against the disabled-operator list, and executes both q_p and q_g with the same normalized executor $\mathrm{NormExec}(\cdot, D)$, which canonicalizes BSON/JSON values and numeric types while preserving semantically meaningful distinctions such as null versus missing fields and retained `_id` values. Parse failures, forbidden operators, execution errors, missing predictions, and typed solver failures are all kept as zero-score per-record rows, so reported averages preserve the full denominator. The metric suite consists of one headline metric and three companions, all derived from the same execution results so they require no extra runs.

a) EXC (headline): bounded column-tolerant execution accuracy: EXC follows the execution-equivalence intuition of database benchmarks, where top-level output column names are not by themselves semantic. A prediction receives $\mathrm{EXC} = 1$ iff it (i) parses, (ii) contains no forbidden operator, (iii) returns the same number of rows as the gold result, and (iv) aligns every gold row with a predicted row whose top-level value multiset contains all gold values with at most $\beta = 2$ surplus top-level values. The surplus bound accounts for benign MongoDB aggregation artifacts, such as a leftover `_id` group key or one helper/sort key, while still rejecting projection-free document dumps. Row order is enforced when the gold pipeline contains order-sensitive root stages (`$sort`, `$limit`, `$skip`, `$setWindowFields`); otherwise rows are matched as multisets. Nested object keys, array order, value identity, row count, and missing values remain strict, so field-name tolerance cannot hide an incorrect answer. EXC is a per-record binary indicator averaged uniformly over the benchmark.

b) EXF$_1$: graded result-set F_1: Because EXC is binary, it cannot distinguish a near-miss from a completely wrong answer. EXF_1 is the order-insensitive multiset F_1 between predicted and gold result rows under the same row identity as EXC (top-level column names cosmetic, nested keys and values significant) but with no surplus tolerance. It grades how close a wrong answer is to the gold result set while collapsing to zero precision on projection-free dumps, so it cannot be gamed by returning extra rows.

c) Execution-outcome decomposition: Every scored record falls into exactly one mutually exclusive bucket: *correct* (the EXC successes), *no submission*, *invalid* (parse failure or forbidden operator), *execution error*, *empty result*, or *wrong rows* (executed but the row multiset or its values disagree, reported as structural near misses, i.e., order-only, row-subset, and row-superset, and as value mismatches). The bucket fractions sum to one, turning the headline gap $1-\mathrm{EXC}$ into an interpretable loss budget that reports *where* along the schema-less reasoning claim systems fail.

d) Claim-axis slices: EXC and EXF_1 are sliced along the benchmark's claim axes of schema-flex category, anti-SQL-transfer level, and database domain (Section II-C), with each slice reported alongside its record count, so an aggregate score cannot hide collapse on a rare native structure such as the 13-record attribute-bag slice.

For the ablation study, where neighboring ladder arms can differ by only a handful of records, we additionally use an

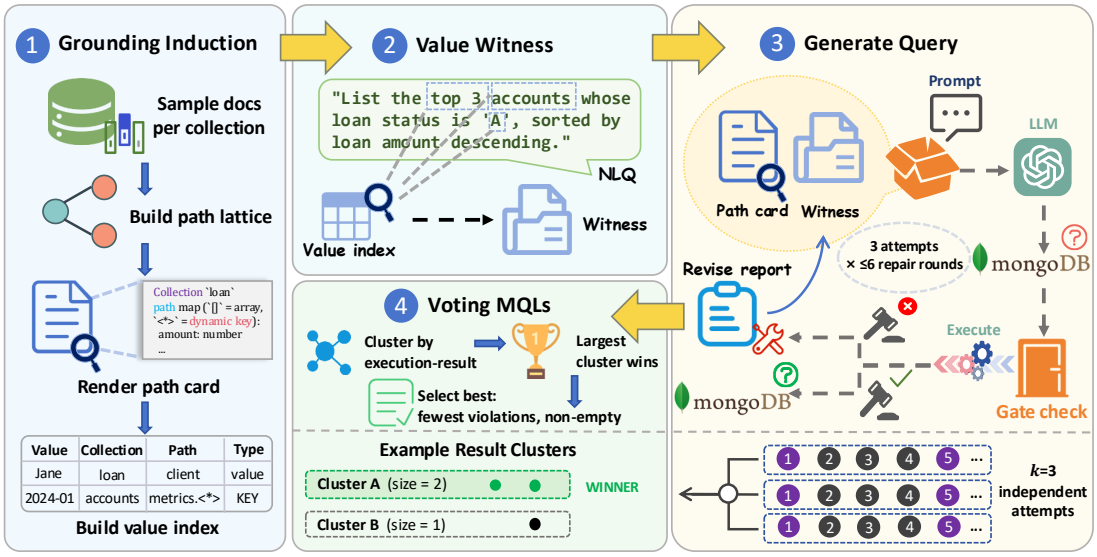

Fig. 4. SAG inference under Schema-as-Data Grounding. SAG first induces a path and value grounding index from stored MongoDB witnesses, then uses grounded value witnesses, gated generation, execution-based repair, and result-cluster voting to select a deterministic MQL prediction.

exact paired McNemar test of each arm's EXC against SAG, so claimed mechanism contributions are separated from noise. We deliberately do not report query-form or naming diagnostics (e.g., exact program match or field-naming agreement): program-form match is near zero for any independently generated query and output-column naming is underdetermined by the request, so neither discriminates between systems.

III. SAG: Schema-as-Data Grounding

SAG addresses Text-to-NoSQL as evidence-constrained program synthesis rather than one-shot query generation. Given a natural-language query and read-only access to a MongoDB database, the goal is to produce a deterministic aggregation pipeline that answers the query while respecting schema-less document structure. This setting differs from NL2SQL inference because the operative schema is not a fixed catalog: fields may be sparse, nested, repeated, dynamically keyed, or visible only after aggregation stages reshape the document stream. SAG therefore treats stored documents as schema evidence. Database contents induce the admissible path space, value evidence, and repair signals that constrain decoding and validation.

A. Overview

Figure 4 shows the end-to-end solver. SAG is organized around a per-database grounding index and a bounded per-query decode–gate–repair procedure. The index is built from bounded witness sampling over the read-only database. It records the collection set, the induced path lattice, dynamic-key maps collapsed to <⋆>, top-level fields, value witnesses, and compact card text that is provided to the decoder. During inference, the model receives only the NLQ, this induced card, and optional value-witness lines; gold programs and benchmark annotations are excluded from the input.

The design has four principles: schema-less structure is treated as data-induced perception, with every admissible field path represented in the induced lattice and dynamic maps exposed through collapsed paths and example keys; value grounding anchors hard NLQ literals to exact stored forms, including dynamic-map keys; deterministic gates check path and value admissibility before execution; and execution feedback serves as a repair gradient, with repaired samples clustered by result equivalence so the final answer is selected in result space rather than by surface form. Unlike ReAct, SAG grounds deterministically before generation and reserves the loop for bounded correction of concrete violations.

B. Problem Formulation and Grounding Objective

Let a MongoDB database be $D = \{C_1, \ldots, C_m\}$, where each collection C_i is a multiset of JSON-like documents rather than a fixed relation. A user utterance is denoted by x, and the target output is a read-only aggregation program $p = (c, \mathbf{s})$ consisting of a base collection $c \in D$ and an ordered stage sequence $\mathbf{s} = (s_1, \ldots, s_T)$. Before decoding, SAG induces a grounding index

$$\mathcal{I}_D = (\mathcal{C}, \mathcal{L}, \mathcal{P}, \mathcal{M}, \mathcal{V}, \mathrm{card}), \tag{1}$$

where $\mathcal{C}$ is the closed collection set, $\mathcal{L}$ is the induced path lattice, $\mathcal{P}$ contains valid path prefixes, $\mathcal{M}$ records dynamic-key maps and example keys, $\mathcal{V}$ is the indexed set of witnessed stored values and dynamic keys, and card is the rendered path card used as the decoder's hypothesis space. A candidate program induces read claims over collections, paths, values, limits, and joins:

$$\Gamma(p) = \Gamma_{\mathrm{coll}}(p) \cup \Gamma_{\mathrm{path}}(p) \cup \Gamma_{\mathrm{value}}(p) \cup \Gamma_{\mathrm{limit}}(p) \cup \Gamma_{\mathrm{join}}(p). \tag{2}$$

The alignment gate implements an admissibility predicate

$$A(p; \mathcal{I}_D, x) = A_{\mathrm{path}}(p; \mathcal{I}_D) \wedge A_{\mathrm{value}}(p; \mathcal{I}_D, x) \wedge A_{\mathrm{limit}}(p, x), \tag{3}$$

where A_{path} checks the selected collection, field references, computed intermediates, dynamic-key accesses, and data-witnessed `$lookup` edges; A_{value} requires hard witnessed literals to appear with exact stored forms on witnessed paths; and A_{limit} enforces explicit top/first/limit requests in the NLQ. Each predicate has a constructive counterpart $V_\bullet$ that returns the set of concrete violations it detects, so that $A_\bullet$ holds exactly when $V_\bullet = \emptyset$; these violation sets are what the repair loop consumes (Algorithm 1). Probe failures are fail-open so that the gate does not reject gold-equivalent programs merely because an auxiliary witness probe is unavailable.

SAG therefore searches over read-only deterministic aggregation programs under admissibility and execution feedback:

$$p^* = \arg \max_{p \in \Pi_{\text{ro}}(D)} \Big[S_\theta(p \mid x, \text{card}, W_x) + \lambda_{\text{gate}} \mathbb{I}\{A(p; \mathcal{I}_D, x)\} + \lambda_{\text{exec}} \mathbb{I}\{\text{Exec}_D(p) \text{ nonempty}\} \Big], \tag{4}$$

where $\Pi_{\text{ro}}(D)$ is the space of read-only aggregation programs over D, W_x denotes the value witnesses extracted by matching literals in x against $\mathcal{V}$, and $E_x \subseteq W_x$ is the enforceable subset of hard witnesses (those with few enough locations to gate on). SAG realizes Eq. 4 procedurally: the LLM proposes structured candidates, gates return repair messages for inadmissible candidates, execution exposes execution errors or empty outputs, and the selected candidate minimizes violations and empty results across bounded repair rounds.

Algorithm 1 summarizes inference. Unlike ordinary agent prompting, structure discovery is not delegated to an in-loop exploratory agent: the complete data-induced card is provided before decoding, and the loop only corrects concrete violations (nonexistent paths, inadmissible joins, wrong literal bindings, missing limits, execution errors, empty results, and leaked identifiers).

C. Grounding Index and Path Card

The grounding index is the schema-as-data representation constructed for each database. SAG samples documents from each collection and recursively walks objects and arrays up to a bounded depth. Each observed node becomes a lattice entry with a dominant type, count, and, for object nodes, a static-document or dynamic-map classification. Dynamic maps are detected from key proliferation and data-like keys, then rendered with the placeholder <∗>; this prevents concrete keys such as dates, codes, and map entries from exhausting the context while still revealing the access pattern.

From the lattice, SAG derives three inference artifacts. The valid-prefix set supports path membership checks; the top-field set distinguishes physical input fields from computed intermediates introduced later in an aggregation pipeline; and the path card renders the entire induced lattice as decoder context. The decoder instruction specifies that paths absent from the induced card should be treated as nonexistent, that related data are usually embedded rather than joined, and that stored values must be copied verbatim unless the question explicitly defines a label mapping.

Algorithm 1: SAG inference under Schema-as-Data Grounding.

Input : User utterance x, read-only database D, policy (k, R)
Output: Aggregation program p or failure diagnostic

$\mathcal{I}_D \leftarrow$ BUILDGROUNDINGINDEX(D);
$(W_x, E_x) \leftarrow$ WITNESSLITERALS$(x, \mathcal{I}_D)$;
$\mathcal{O} \leftarrow \emptyset$;
for $i \leftarrow 1$ **to** k **do**
  $M \leftarrow [$SYSTEMPROMPT$(\mathcal{I}_D),$ QUESTION$(x, W_x)]$;
  $\mathcal{B} \leftarrow \emptyset$;
  for $r \leftarrow 1$ **to** R **do**
    $T \leftarrow \perp$;
    $p \leftarrow$ DECODESTRUCTURED$(M, \mathcal{C})$;
    $F \leftarrow V_{\text{path}}(p, \mathcal{I}_D) \cup V_{\text{value}}(p, E_x) \cup V_{\text{limit}}(p, x)$;
    if $F = \emptyset$ **then**
      $T \leftarrow$ EXECUTEREADONLY(p, D);
      if T *is empty* **then**
        $F \leftarrow \{$BISECTEMPTY$(p, D, \mathcal{I}_D)\}$
      else if T *leaks synthetic* `_id` **then**
        $F \leftarrow \{$SUPPRESSID$\}$
    $\mathcal{B} \leftarrow \mathcal{B} \cup \{(p, F, T)\}$;
    if $F = \emptyset$ **then**
      break
    $M \leftarrow M \cup [$CANDIDATE$(p),$ REPAIRFEEDBACK$(F)]$;
  $\mathcal{O} \leftarrow \mathcal{O} \cup \{$BESTBYVIOLATIONS$(\mathcal{B})\}$;
if $\mathcal{O} = \emptyset$ **then**
  return FAIL*(no valid candidate)*
return LARGESTRESULTCLUSTER$(\mathcal{O})$

The card is intentionally stronger than a name-level schema summary: induced from the same read-only database the generated query executes on, it includes nested array paths, exposes sparse and presence-wrapped fields when observed, and represents dynamic-key structures compactly.

D. Value Witnesses and Alignment Gates

SAG complements path grounding with value-witness anchoring: quoted strings, dates, all-caps codes, and selected title-case phrases from the NLQ are normalized and looked up in the value index. A witness records where the term occurs, whether it is a stored value or a dynamic key, and the exact stored form; hard witnesses with few locations are enforceable, while soft witnesses inform the model but do not block candidates.

The deterministic gate has two main sides. A_{path} checks that the selected collection exists, every outer-pipeline field reference resolves to an induced path or valid computed intermediate, and each `$lookup` targets a real collection through a data-witnessed join edge. If a join edge cannot be probed safely, the gate fails open, but nonexistent collections and missing foreign fields remain violations. A_{value} checks comparison-position strings against hard witnesses so that a model cannot observe a literal in one path and silently filter a different path or casing.

A third contract enforces explicit row limits: a top-N, first-N, or at-most-N request must carry the corresponding

`$limit`. Together these checks turn common failure modes into model-facing feedback: wrong collection names, hallucinated paths, unnecessary joins over embedded data, misbound constants, and missing limit semantics.

E. Execution Repair and Result Consistency

After a candidate passes the alignment gate, SAG scans for disabled operators and executes the pipeline against the read-only world. Execution is not used as a gold-answer oracle; it is a validity and repair signal. Execution errors are returned as concise feedback. Empty results trigger prefix bisection: SAG executes successive pipeline prefixes with `$count` to locate the first stage that collapses the stream to zero rows or errors, and, when possible, reports actual distinct values at filtered paths. This converts an uninformative "0 rows" outcome into a localized, data-grounded repair gradient.

SAG also enforces an output contract for internal identifiers: a pipeline that leaks synthetic `_id` values is repaired unless the question asks for them. Repair feedback includes the previous candidate and its concrete violations and requests a corrected structured candidate; within an attempt, SAG retains the candidate with the fewest violations, preferring non-empty executions and later rounds.

SAG uses $k = 3$ attempts in its full configuration: final executions are clustered by order-insensitive result equivalence, the largest cluster supplies the prediction, and ties prefer fewer violations and non-empty results. Selection happens in result space because MongoDB programs with different surface forms can compute the same answer while plausible pipelines diverge after an unwind, projection, or group. If all attempts fail, SAG reports a failure diagnostic rather than an unsupported query.

IV. Experiments and Analysis

This section evaluates SAG on the TEND release, compares it with controlled information-channel baselines, measures SAG mechanism ablations, and reports EXC together with diagnostic metrics. The results show that TEND is a difficult benchmark even for a strong reasoning model: the best system remains below 36% mean EXC, while SAG obtains the highest mean accuracy, the highest graded result quality, and the cleanest executable behavior under the same model and evaluator.

A. Experimental Setup

1) Dataset: Experiments use the public *tend-native-mongodb-v1* release (1,210 execution-distinct tasks over 11 databases; Section II-C). All systems are evaluated on the identical task set against the same frozen MongoDB witness data, with missing predictions, execution errors, and typed failures retained in the denominator as zero-score rows.

2) Models: All compared systems share the same backbone LLM, decoding configuration, and evaluator; they differ only in their *information channel*, that is, in what each method is allowed to observe about the schema-less MongoDB database, and in whether that channel is static (one-shot) or interactive (exploration or repair). No method receives the gold MQL, the canonical-form set, or any release annotation. The compared methods are:

- **NLQ-only Direct.** A channel-purity floor and contamination probe: only the natural-language question, with no schema, samples, or database access; the pipeline is emitted in one shot. Non-trivial accuracy here would indicate memorization rather than grounding.
- **Schema Direct.** The question plus a name-level structural summary of the database (collection names and document field names, no stored values or samples), isolating how far name-level structure alone carries.
- **Sampled-doc Direct.** The question plus three raw documents per collection, from which collections, field paths, and value formats must be inferred in one shot.
- **Data-rich Direct.** Identical except for a larger sample budget (eight documents per collection), separating sample size from mechanism.
- **SQL Pivot.** The same channel as Sampled-doc Direct, but the model first organizes the intent as a self-generated relational SQL sketch and then translates it into MQL, with no gold SQL, relational source schema, external converter, or execution feedback. This arm measures whether a relational intermediate helps or hurts document-native reasoning (Section IV).
- **ReAct.** An agentic exploration baseline and the apples-to-apples counterpart of SAG: given the same collection-name list SAG receives, it self-paces a bounded thought–action–observation loop of read-only actions with *raw* first-five-row observations. Budget exhaustion without a submitted pipeline is a typed failure, never converted into a forced answer. Unlike SAG it receives no induced path card, value witnesses, alignment gates, or repair feedback.
- **SAG (ours).** The proposed solver in its full configuration (Section III): induced path-lattice card and value index from a bounded witness sample, value witnesses for question literals, the joint path-and-value admissibility gate with the limit contract, at most six repair rounds with prefix-bisection feedback on empty results, and three-sample result-space consistency. All probes are read-only; execution is repair evidence only, never a correctness oracle.

All non-agentic baselines see only sanitized public record fields and emit machine-checkable disclosure fields stating exactly which channel they observed; ReAct's raw-row visibility is likewise declared in its disclosure.

3) Evaluation Metrics: The headline execution metric is EXC, the bounded column-tolerant execution accuracy defined in Section II-D. We additionally report the graded result-set F_1 (EXF_1) and a mutually exclusive execution-outcome decomposition. In the compact tables, NoSub denotes missing or typed no-submission failures, Exec denotes MongoDB execution errors, Empty denotes executable pipelines that return no rows, Struct. aggregates order-only, row-subset, and row-superset near misses, and Value denotes value-mismatch

TABLE III
MAIN COMPARISON ON TEND. BEST ACCURACY VALUES ARE BOLDED.

Method	Accuracy		Outcome decomposition (%)				
	EXC	EXF_1	NoSub	Exec	Empty	Struct.	Value
NLQ-only Direct	0.2	0.2	8.2	1.1	89.4	0.0	1.2
Schema Direct	0.2	0.4	6.3	15.3	69.3	0.1	8.8
Sampled-doc Direct	28.6	29.4	2.1	4.7	7.2	1.6	55.7
Data-rich Direct	28.9	29.8	4.8	3.1	3.8	1.6	57.7
SQL Pivot	25.6	26.2	1.5	4.9	8.8	1.4	57.8
ReAct	30.8	34.6	9.0	0.2	0.3	2.7	56.9
SAG (Ours)	**35.8**	**40.5**	0.0	0.0	0.5	2.2	61.5

failures. Invalid candidates are retained in the denominator; they are 0.0% for all headline rows in this snapshot and are therefore omitted from the compact tables. The bucket fractions sum to one with EXC, and ablation significance statements use an exact paired McNemar test of each arm's EXC against SAG.

4) Implementation Details: All systems use the same backbone LLM, DeepSeek-V4-Flash, a reasoning model accessed through an OpenAI-compatible API with reasoning effort *max*, temperature 0.0, and no output-token cap (a cap truncates long reasoning traces and artificially inflates no-submission failures); the identical client, decoding configuration, and evaluator make reported differences attributable to the information channel and mechanism alone. SAG runs with k=3 attempts, at most six repair rounds per attempt, a 400-document witness sample per collection, a 400-entry path card, and a 20-second per-query execution timeout; ReAct receives 50 interaction steps per task. All probing and execution are read-only against the frozen release witness data in a live MongoDB instance; no system writes to the database or observes evaluation output.

B. Performance Comparison

Table III reports the main comparison. **The channel-purity floors show that TEND can only be solved by grounding.** NLQ-only Direct and Schema Direct both reach 0.2% EXC, so the release is not recoverable from language priors (a clean contamination probe) and name-level structure is nearly worthless: 69.3% of Schema Direct pipelines execute to empty results because collection and field names reveal neither dynamic keys, populated paths, nor stored value forms. Even with grounding the benchmark stays hard. The best system solves 35.8%, no method exceeds 54.5% on any database, and for every value-bearing channel the dominant failure bucket is wrong rows, that is, pipelines that parse, execute, and return plausible but incorrect results, the failure regime only execution-result evaluation exposes.

More data without mechanism saturates immediately; SAG's gain is mechanism. Nearly tripling the one-shot sample (Sampled-doc to Data-rich Direct) buys +0.3 points, and 50 steps of interactive exploration (ReAct) buy +1.9 more at the panel's highest no-submission rate (9.0%, budget exhaustion). SAG reaches 35.8% EXC and 40.5 EXF_1, a gain of +5.0 EXC over ReAct and +6.9 over the strongest one-shot channel, while replacing open-ended exploration with bounded repair (at most six rounds, averaging close to one). Its EXF_1-EXC gap (4.7 points, the largest in the panel) shows that even SAG's misses are near-misses in result space rather than structural derailments. The interaction economics compound the accuracy gap: in our development runs the SAG arms averaged 1.1 to 1.2 repair rounds per attempt against 13 to 15 consumed ReAct steps per task, so grounding before decoding is not only more accurate but roughly an order of magnitude cheaper in model interactions.

TABLE IV
EXC (%) BY CLAIM-AXIS SLICE.

Method	Schema-flex category					Anti-SQL	
	DynKey (1013)	Event (130)	Poly (36)	Missing (18)	AttrBag (13)	Med. (71)	Strong (1138)
Data-rich Direct	29.1	27.7	25.0	44.4	15.4	23.9	29.2
SQL Pivot	26.0	22.3	33.3	33.3	0.0	14.1	26.3
ReAct	28.5	**45.4**	**41.7**	44.4	15.4	**36.6**	30.4
SAG (Ours)	**35.5**	38.5	22.2	44.4	**53.8**	29.6	**36.1**

DynKey = dynamic-key maps, Event = nested event streams, Poly = polymorphic shapes, Missing = missing-vs-present, AttrBag = attribute bags; record counts in parentheses. The single-record *weak* transfer slice is omitted.

SAG eliminates the mechanical failure surface. It is the only system with zero no-submission, invalid, or execution-error rows; its residual loss is concentrated in the semantic wrong-rows bucket where the real headroom lies. At the database grain (all systems run one fixed configuration across the eleven databases, with no per-database tuning), SAG is best or tied-best on seven of eleven, with its largest leads where document-native structure dominates: 50.9% versus 36.4% for the best baseline on `toxicology`, 50.0% versus 41.8% on `european_football_2`, and 33.6% versus 20.9% on `superhero`, while matching the best baseline on `financial` (54.5%). Per-database EXC ranges from 17.3% (`formula_1`) to 54.5%, ReAct retains an edge on `formula_1` and `thrombosis_prediction`, and no method dominates every database, the intended behavior of a benchmark whose DataWorlds stress different combinations of dynamic keys, embedded evidence, sparse paths, and aggregation shape.

C. Where the Difficulty Comes From

Table IV resolves the gap along the claim axes. **SAG's margin concentrates exactly where schema-less structure carries the query**: +6.4 points over the best baseline on the dominant dynamic-key slice (1,013 of 1,210 records), 53.8% versus 15.4% on attribute bags, and +5.7 on the strong anti-SQL-transfer slice (94% of the release). These are the slices where a name-level catalog is least informative, because the query-bearing structure lives in object keys, array contents, embedded records, and stored values. ReAct stays competitive on the two smallest structural families, nested event streams (n=130) and polymorphic shapes (n=36), where iteratively probing a handful of co-existing document variants can locally substitute for global lattice induction.

TABLE V
ABLATION OF SAG.

Method	Accuracy		Failures (%)				p
	EXC	EXF_1	NoSub	Empty	Struct.	Value	vs. SAG
Cumulative mechanism ladder (full panel, 1,210 tasks)							
Path Card	31.3	35.0	4.7	5.5	2.3	56.2	< 0.001
Gated Repair	34.8	38.5	0.0	1.2	1.8	62.1	0.353
Value/Bisection Repair	33.6	36.8	0.2	0.4	1.7	64.2	0.027
SAG (Ours)	**35.8**	**40.5**	0.0	0.5	2.2	61.5	–
Component knockouts[a]							
SAG (same batch)	41.8	43.0	0.0	0.0	1.4	56.8	–
w/o alignment gate	45.0	47.1	0.0	0.5	0.9	53.6	–
w/o value witnesses	44.1	46.4	0.0	0.9	1.9	53.2	–
w/o bisection feedback	45.5	48.0	0.0	0.9	1.4	52.3	–
top-level card	6.4	7.2	0.9	29.5	0.5	62.7	–
w/o dynamic-key collapse	37.7	39.0	0.0	0.0	0.9	61.4	–

[a] Each knockout removes exactly one mechanism from the full solver, measured on `financial+superhero`; the reference row is the same-batch full solver.

The headline is not a metric artifact. All 1,210 gold pipelines self-score EXC=1, an independent β-sweep selects the frozen surplus bound β=2, and counterfactual witness-injection flips only 3 of 201 SAG-correct verdicts, so the reported gains are not artifacts of empty outputs, helper-column tolerance, or fragile database instances.

D. Ablation Study

We ablate SAG through a cumulative mechanism ladder whose settings are named by the strongest capability available in each arm. **Path Card** gives the decoder the induced path card but performs a single ungated generation. **Gated Repair** adds path admissibility checking and execution-grounded repair with plain empty-result feedback. **Value/Bisection Repair** further adds value witnesses, value alignment, the limit contract, and prefix-bisection feedback, but keeps a single final attempt. **SAG** is the complete proposed solver, adding three-way result-consistency clustering. Table V uses the same execution-result vocabulary as the main comparison table, so each ablation result is read as both an accuracy change and a failure-mode redistribution. Because adjacent ladder arms can differ by only a handful of records, significance statements use an exact paired McNemar test of each arm's per-record EXC against SAG rather than raw deltas alone.

The ladder reads as the construction of the solver. **The induced card alone already beats every baseline.** A single ungated decode conditioned on the data-induced lattice reaches 31.3% EXC, above the 50-step interactive agent (30.8%) and every one-shot channel, at one model call per task. In a schema-less world, structure discovery behaves like perception: induced completely from stored documents up front, it is worth more than an entire exploration loop re-deriving it per task. **Gating and repair convert the remaining mechanical tax into accuracy**, adding +3.5 points (the largest single rung) while eliminating no-submissions and cutting empty executions from 5.5% to 1.2%. The gate does this without endangering recall: it accepts all 1,210 gold pipelines across the 11 databases with zero false positives, so by construction it can only remove inadmissible candidates, never a gold-equivalent answer. **The value contract needs consistency to pay off.** Enforced under a single final attempt (Value/Bisection Repair), it is the one non-monotone rung (33.6%), because a stricter contract applied to a single derivation occasionally repairs a near-miss into a different wrong program with no independent derivation to fall back on. Full SAG resolves exactly this tension: k=3 result-space clustering gives the contract multiple independent derivations to arbitrate among, recovering +2.2 points (p=0.027) and finishing +4.5 above the card alone (p<0.001). The closest rung, Gated Repair, is not separable from full SAG at pooled significance (p=0.353); its residual gap concentrates on individual DataWorlds (+5.4 points on `toxicology`, +4.6 on `formula_1`) and is averaged down by pooled testing.

Component knockouts confirm where the mechanism lives. The lower block of Table V removes exactly one mechanism at a time from the full solver. The card-construction knockouts are decisive: a top-level-fields card collapses the solver from 41.8% to 6.4% EXC with 29.5% empty executions, reproducing the Schema-Direct failure profile because nested and dynamic-key paths vanish from the hypothesis space, and disabling dynamic-key collapse, which floods the card with concrete stored keys until the entry cap truncates coverage, costs 4.1 points. Both card knockouts also behave consistently across the two databases (6.4% on each for the top-level card; 47.3% and 28.2% without collapse), so schema-as-data grounding stands or falls with the completeness of the induced lattice, and collapse is what keeps completeness affordable inside a bounded card. The three guard knockouts (gate, value witnesses, bisection feedback) each shift the subset mean by at most 3.7 points (at most eight records), within run-to-run variation at this scale; the same full configuration scores 33.6% on `superhero` in the canonical panel but 29.1% in the knockout batch under k=3 sampling, and the guards overlap by construction, since an unwitnessed literal that the value contract would flag typically also drives an empty execution that bisection then localizes. Their measured contribution is panel-level rather than subset-mean: zero mechanical failures in Table III, the zero-false-positive safety of the gate, and the contract that the consistency stage needs, as the following control shows.

Consistency is a selection rule, not a sampling bonus. Wrapping the strongest one-shot baseline in exactly SAG's k=3 result-clustering rule (on the `financial`/`superhero`/`toxicology` subset) is flat to negative, 36.0% to 35.5% mean EXC, and on `superhero` it amplifies a systematic error (20.9% to 14.5%): without admissibility gating, result-space voting arbitrates among unconstrained guesses and selects the consistent-but-wrong cluster. On the same three databases full SAG reaches 46.3% EXC, +10.8 points over the compute-matched wrap. Consistency pays only on top of gated, witness-grounded candidates; it is a selection rule over admissible programs, not a substitute for grounding.

TABLE VI
CASE STUDY ON THE THROMBOSIS_PREDICTION DATAWORLD.

NLQ. *Show the top 100 measurement-code groups that have readings* ***for 1987****. Group by supplemental reference range and year, count documents in each group, and sort primarily by document count descending.*
Witness structure. `measurement_code_bags` (25 docs): `{ code, bag_state: { supplemental_reference_range, ··· }, values_by_year: { "1981": { reading_count, readings: [···] }, "1982": ··· } }`. Per-year readings sit under *dynamic year keys* (≈12 keys per document); only some documents carry a `"1987"` entry, and the reference range is stored as an explicit `"missing"` sentinel.

System	Generated pipeline (verbatim, one stage per line)	Executed result / EXC
Gold *reference*	`{ $addFields: { category: { $ifNull: ["$bag_state.supplemental_reference_range", "missing"] } } }` `{ $project: { dynamic_entries: { $objectToArray: "$values_by_year" } } }` `{ $unwind: "$dynamic_entries" }` `{ $match: { "dynamic_entries.k": "1987" } }` ← *the 1987 restriction is applied before grouping* `{ $group: { _id: { category, entry_key }, document_count: { $sum: 1 } } }` `{ $project: ··· } { $sort: { document_count: -1, ··· } } { $limit: 100 }`	**1 group:** `[{ document_count: 12, category: "missing", entry_key: "1987" }]` *the reference answer*
SQL Pivot *sampled docs → SQL sketch → MQL*	`{ $match: { "values_by_year.1987": { $exists: true } } }` ← *existence check only, not a restriction* `{ $addFields: { year_entries: { $objectToArray: "$values_by_year" } } }` `{ $unwind: "$year_entries" }` ← *all 19 year keys re-expanded; the 1987 restriction is lost* `{ $group: { _id: { supplemental_reference_range, year: "$year_entries.k" }, doc_count: { $sum: 1 } } }` ← *every stored year becomes a group; the compound _id leaks* `{ $sort: { doc_count: -1 } } { $limit: 100 }`	✗ **19 groups:** `[{ _id: { supplemental_reference_range: "missing", year: "1988" }, doc_count: 12 }, { _id: { ···, year: "1987" }, doc_count: 12 }, ···]` *one group per stored year; plausible rows, wrong content*
ReAct *collection list + interactive probing*	`{ $match: { "values_by_year.1987": { $exists: true } } }` `{ $project: { range: "$bag_state.supplemental_reference_range", years: { $objectToArray: "$values_by_year" } } }` `{ $unwind: "$years" }` ← *probing found the map, then the same relational habit* `{ $group: { _id: { range, year: "$years.k" }, count: { $sum: 1 } } }` `{ $sort: { count: -1, ··· } } { $limit: 100 } { $project: { _id: 0, ··· } }`	✗ **19 groups:** `[{ supplemental_reference_range: "missing", year: "1987", count: 12 }, { ···, year: "1988", count: 12 }, ···]` *same failure as SQL Pivot*
SAG (ours) *path card + value witnesses*	`{ $match: { "values_by_year.1987.reading_count": { $gt: 0 } } }` ← *the literal 1987 is addressed as a stored map key; no flattening at all* `{ $group: { _id: { reference_range, year: "1987" }, count: { $sum: 1 } } }` `{ $sort: { count: -1 } } { $limit: 100 } { $project: { _id: 0, ··· } }`	✓ **1 group:** `[{ count: 12, reference_range: "missing", year: "1987" }]` *matches the gold result exactly*

Notes. Verbatim pipelines generated for the same NLQ, one stage per line (only tail boilerplate is elided, ···). Red marks the decisive wrong stage and green the decisive correct one, with the reason annotated inline (←). All four pipelines execute without error; the outcome column reports the executed result and the EXC verdict (✓/✗).

E. Case Study

Table VI contrasts the pipelines generated for one task over the `thrombosis_prediction` DataWorld, where per-year readings sit in a dynamic-key map `values_by_year` and the NLQ asks for groups with readings *for 1987*. Both baselines discover the map; **SQL Pivot** checks `values_by_year.1987` for existence and **ReAct** probes the collection interactively. Both then fall back on the same relational habit: flatten the map with `$objectToArray` and `$unwind`, group by the re-expanded year column, and silently drop the 1987 restriction, returning all 19 (range, year) groups in place of the single gold group. The queries parse, execute, and look plausible; only execution-result comparison exposes them. **SAG** instead resolves the literal "1987" through its value index to a witnessed dynamic-map key, addresses `values_by_year.1987` directly, and matches the gold result exactly. The case illustrates the failure mode that separates systems on TEND: not failing to *find* MongoDB-native structure, but reasoning about it relationally once found. SAG's card renders the map as a first-class hypothesis (`values_by_year.<*>`), its witnesses bind literals to stored keys, and its gate rejects unwitnessed paths and literal forms, keeping the pipeline inside the document-native representation of the constraint.

F. Analysis of Relational-Pivot Transfer

SQL Pivot isolates the relational intermediate under a controlled channel: it shares its sampled-document input with Sampled-doc Direct and self-generates a SQL sketch before translating it to MQL, with no gold SQL, relational catalog, external converter [17]–[19], or execution feedback, so the pair measures representation mismatch rather than converter coverage. **The intermediate is a net loss**: 25.6% versus 28.6% EXC, with more empty executions (8.8% vs. 7.2%). The losses concentrate exactly where document structure is query-bearing (Table IV): 14.1% on the medium anti-transfer slice and 0.0% on attribute bags, the weakest value-bearing cells in the panel. The mechanism is the one the case study shows verbatim: the sketch expresses document evidence as imagined tables, joins, and globally named columns, and the flatten-then-group translation silently drops dynamic-key restrictions and leaks compound group keys. A self-generated relational intermediate is not merely unhelpful on TEND; it is actively harmful, which

is precisely the anti-SQL-transfer property the benchmark is designed to measure.

V. Related Work

Our work intersects NoSQL data management, natural-language query generation, and execution-centered benchmark design. We position Text-to-NoSQL as a broader problem family whose MongoDB-native instantiation requires executable query synthesis over schema-less document stores.

A. NoSQL Databases

Document stores such as MongoDB replace fixed table-column catalogs with collections of nested documents, embedded arrays, optional fields, and evolving key sets, which weakens the static schema assumptions of most natural-language database benchmarks. Database research on NoSQL systems has studied scalability and reliability in distributed environments [8], query processing for large-scale data management [20], and security and privacy [21]; these works address system-level concerns. TEND is complementary: it studies how natural-language systems should generate executable MQL when the relevant schema is partly implicit in the document instances, so a benchmark must test inference of field paths, array structure, dynamic-key usage, and result contracts from a concrete database world, not only MongoDB syntax.

A small set of MongoDB-querying resources is better viewed as SQL-to-MQL transfer context, where MongoDB programs are derived from relational schemas and gold SQL. DocSpider [15] is the most direct example: it migrates the Spider databases into MongoDB with one collection per table and flat documents, then uses LLMs to translate Spider's gold SQL into 4,663 MQL queries gated by execution-equivalence checks. The resulting databases contain no nested, optional, polymorphic, or dynamic-key structure (its own nested-query evaluation falls back to a separate 20-query sample over an external Airbnb collection), so it measures SQL-to-MQL syntax transfer over relational mirrors (Table II). TEND's claim is complementary and concerns MongoDB-native DataWorld design, where collection boundaries, embedded records, nesting, dynamic keys, optional paths, sparse fields, polymorphic shapes, and witness values are benchmark variables.

Recent NL-to-NoSQL work has also begun to examine language coverage: MultiTEND [22] studies multilingual natural-language-to-NoSQL translation with multilingual linking mechanisms, a direction complementary to TEND's MongoDB-native DataWorld focus.

B. Text-to-SQL

Text-to-SQL connects natural language with relational database systems and has been studied extensively in both the database and NLP communities, from rule-based and sequence-to-sequence methods [23]–[28], through attention, graph representations, and constrained decoding [29]–[38], to the current LLM-centric paradigm [39]–[45], with benchmarks such as WikiSQL [11], Spider [12], KaggleDBQA [13], and BIRD [10] making cross-domain relational evaluation standard. Text-to-SQL remains an important baseline family for Text-to-NoSQL because SQL-derived transfer exposes how much relational structure carries over. However, relational benchmarks rely on static, globally defined schemas, whereas document-store Text-to-NoSQL requires reasoning over nested arrays, sparse fields, dynamic keys, optional paths, and polymorphic structures; relational-pivot and SQL-to-NoSQL channels are therefore useful comparisons but cannot define MongoDB-native benchmark semantics.

C. Natural Language Interfaces for Data Systems

Natural-language interfaces to data systems extend beyond relational SQL: query generation for graph databases and the Overpass geographic database [46], [47], speech-driven querying [48], [49], and broader NLIDS surveys [50] motivate non-relational access, but these data models do not test schema-less document reasoning over executable aggregation pipelines. Adjacent visualization-facing interfaces study robust, free-form, conversational, and multi-agent text-to-visualization [51]–[56]; they share the goal of lowering data-access barriers, but their target artifacts are visualizations or answers over them rather than executable document-store queries. SM3-Text-to-Query [16] is the strongest multi-model resource, representing synthetic medical data across PostgreSQL, MongoDB, Neo4j, and GraphDB and evaluating SQL, MQL, Cypher, and SPARQL; we treat it as complementary, since its contribution is cross-model comparison over a controlled source, whereas TEND isolates MongoDB-native benchmark design with expert-authored DataWorlds, schema-less query-bearing structure, frozen witness execution, and program-diversity control.

VI. Conclusion

This study formalizes the document-oriented branch of Text-to-NoSQL as executable query synthesis over schema-less MongoDB document stores. We develop TEND, an execution-verified benchmark that makes document structure, dynamic keys, sparse paths, and witness values part of the benchmark semantics; SAG, a schema-as-data grounding solver that induces structure from witness documents before bounded generation and repair; and an execution-centered evaluation protocol with EXC, EXF_1, and a mutually exclusive outcome decomposition for failure attribution. Extending the grounding index with targeted probes for polymorphic variants and event streams, where agentic exploration remains competitive, is the most direct avenue for further gains. TEND, the SAG implementation, and the evaluation tooling are publicly released to support reproducible progress on Text-to-NoSQL. We hope these contributions together establish schema-less document reasoning as a measurable and improvable dimension of natural-language data access.

Acknowledgment

We sincerely thank the anonymous reviewers for their valuable comments and constructive suggestions, which helped improve this paper.

AI-Generated Content Acknowledgement

AI coding assistants (Anthropic's Claude, accessed through the Claude Code agent) were used to help implement the experiment code of this work, including the evaluation harness, the baseline and ablation runners, and the result-aggregation scripts described in Section IV. All AI-assisted code was reviewed, tested, and validated by the authors, and all reported experimental results were produced by executing this validated code against the frozen release data. The text, figures, and tables of this article were authored by the authors; no AI-generated text or figures appear in the article.

References

[1] J. Lu, Z. Qin, C. Zhang, H. Zhang, Y. Song, and R. C.-W. Wong, "Querycraft: A natural language-driven nosql database querying system powered by large language models," *Proc. VLDB Endow.*, 2026, accepted to the VLDB 2026 Demonstrations Track. [Online]. Available: https://vldb.org/2026/demonstrations.html

[2] H. Vera-Olivera, R. Guo, R. C. Huacarpuma, A. P. B. Da Silva, A. M. Mariano, and M. Holanda, "Data modeling and nosql databases-a systematic mapping review," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–26, 2021. [Online]. Available: https://doi.org/10.1145/3457608

[3] F. Gessert and N. Ritter, "Scalable data management: Nosql data stores in research and practice," in *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 2016, pp. 1420–1423. [Online]. Available: https://doi.org/10.1109/ICDE.2016.7498360

[4] A. Davoudian, L. Chen, and M. Liu, "A survey on nosql stores," *ACM Computing Surveys (CSUR)*, vol. 51, no. 2, pp. 1–43, 2018. [Online]. Available: https://doi.org/10.1145/3158661

[5] J. Han, H. E, G. Le, and J. Du, "Survey on nosql database," in *2011 6th International Conference on Pervasive Computing and Applications*, 2011, pp. 363–366.

[6] D. Ganesh Chandra, "Base analysis of nosql database," *Future Gener. Comput. Syst.*, vol. 52, no. C, p. 13–21, nov 2015. [Online]. Available: https://doi.org/10.1016/j.future.2015.05.003

[7] Y. Li and S. Manoharan, "A performance comparison of sql and nosql databases," in *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, 2013, pp. 15–19.

[8] R. Cattell, "Scalable sql and nosql data stores," *SIGMOD Rec.*, vol. 39, no. 4, p. 12–27, may 2011. [Online]. Available: https://doi.org/10.1145/1978915.1978919

[9] DB-Engines, "DB-Engines Ranking of Document Stores," https://db-engines.com/en/ranking/document%2Bstore, accessed: 2026-06-11.

[10] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, X. Zhou, M. Chenhao, G. Li, K. Chang, F. Huang, R. Cheng, and Y. Li, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 42 330–42 357. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/83fc8fab1710363050bbd1d4b8cc0021-Paper-Datasets_and_Benchmarks.pdf

[11] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017.

[12] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman, Z. Zhang, and D. Radev, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 3911–3921. [Online]. Available: https://aclanthology.org/D18-1425

[13] C.-H. Lee, O. Polozov, and M. Richardson, "KaggleDBQA: Realistic evaluation of text-to-SQL parsers," in *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, C. Zong, F. Xia, W. Li, and R. Navigli, Eds. Online: Association for Computational Linguistics, Aug. 2021, pp. 2261–2273. [Online]. Available: https://aclanthology.org/2021.acl-long.176

[14] F. Lei, J. Chen, Y. Ye, R. Cao, D. Shin, H. Su, Z. Suo, H. Gao, W. Hu, P. Yin, V. Zhong, C. Xiong, R. Sun, Q. Liu, S. Wang, and T. Yu, "Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows," in *International Conference on Learning Representations*, 2025, iCLR 2025 Oral. [Online]. Available: https://openreview.net/forum?id=XmProj9cPs

[15] A. G. Özer, R. F. Cekinel, I. H. Toroslu, and P. Karagoz, "Docspider: a dataset of cross-domain natural language querying for mongodb," *Natural Language Processing*, vol. 31, no. 6, pp. 1367–1398, 2025.

[16] S. Sivasubramaniam, C. Osei-Akoto, Y. Zhang, K. Stockinger, and J. Fürst, "SM3-Text-to-Query: Synthetic multi-model medical text-to-query benchmark," *Advances in Neural Information Processing Systems*, vol. 37, pp. 88 627–88 663, 2024. [Online]. Available: https://openreview.net/forum?id=Pm0UzCehgB

[17] Site24x7, "Sql to mongodb query converter," https://www.site24x7.com/tools/sql-to-mongodb.html, 2023, accessed: 2025-1-11.

[18] JavaInUse, "Sql to mongodb query converter," https://www.javainuse.com/sql2mongo, 2023, accessed: 2025-1-11.

[19] vincentrussell, "Sql to mongodb query converter," https://github.com/vincentrussell/sql-to-mongo-db-query-converter.git, 2024, accessed: 2025-1-16.

[20] D. Mahajan, C. Blakeney, and Z. Zong, "Improving the energy efficiency of relational and nosql databases via query optimizations," *Sustainable Computing: Informatics and Systems*, vol. 22, pp. 120–133, 2019. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2210537918301112

[21] L. Okman, N. Gal-Oz, Y. Gonen, E. Gudes, and J. Abramov, "Security issues in nosql databases," in *Proceedings of the 2011IEEE 10th International Conference on Trust, Security and Privacy in Computing and Communications*, ser. TRUSTCOM '11. USA: IEEE Computer Society, 2011, p. 541–547. [Online]. Available: https://doi.org/10.1109/TrustCom.2011.70

[22] Z. Qin, Y. Song, J. Lu, Y. Song, S. Li, and C. J. Zhang, "MultiTEND: A multilingual benchmark for natural language to NoSQL query translation," in *Findings of the Association for Computational Linguistics: ACL 2025*, W. Che, J. Nabende, E. Shutova, and M. T. Pilehvar, Eds. Vienna, Austria: Association for Computational Linguistics, Jul. 2025, pp. 24 632–24 657. [Online]. Available: https://aclanthology.org/2025.findings-acl.1265/

[23] J. Sen, C. Lei, A. Quamar, F. Özcan, V. Efthymiou, A. Dalmia, G. Stager, A. Mittal, D. Saha, and K. Sankaranarayanan, "Athena++: natural language querying for complex nested sql queries," *Proc. VLDB Endow.*, vol. 13, no. 12, p. 2747–2759, jul 2020. [Online]. Available: https://doi.org/10.14778/3407790.3407858

[24] A. Quamar, V. Efthymiou, C. Lei, and F. Özcan, "Natural language interfaces to data," *Foundations and Trends® in Databases*, vol. 11, no. 4, pp. 319–414, 2022. [Online]. Available: http://dx.doi.org/10.1561/1900000078

[25] C. Baik, Z. Jin, M. Cafarella, and H. V. Jagadish, "Duoquest: A dual-specification system for expressive sql queries," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 2319–2329. [Online]. Available: https://doi.org/10.1145/3318464.3389776

[26] J. Qi, J. Tang, Z. He, X. Wan, Y. Cheng, C. Zhou, X. Wang, Q. Zhang, and Z. Lin, "RASAT: Integrating relational structures into pretrained Seq2Seq model for text-to-SQL," in *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Y. Goldberg, Z. Kozareva, and Y. Zhang, Eds. Abu Dhabi, United Arab Emirates: Association for Computational Linguistics, Dec. 2022, pp. 3215–3229. [Online]. Available: https://aclanthology.org/2022.emnlp-main.211

[27] O. Popescu, I. Manotas, N. P. A. Vo, H. Yeo, E. Khorashani, and V. Sheinin, "Addressing limitations of encoder-decoder based approach to text-to-SQL," in *Proceedings of the 29th International Conference on Computational Linguistics*, N. Calzolari, C.-R. Huang, H. Kim, J. Pustejovsky, L. Wanner, K.-S. Choi, P.-M. Ryu, H.-H. Chen, L. Donatelli, H. Ji, S. Kurohashi, P. Paggio, N. Xue, S. Kim, Y. Hahm, Z. He, T. K. Lee, E. Santus, F. Bond, and S.-H. Na, Eds. Gyeongju, Republic of Korea: International Committee on Computational Linguistics, Oct. 2022, pp. 1593–1603. [Online]. Available: https://aclanthology.org/2022.coling-1.137

[28] R. Cai, B. Xu, Z. Zhang, X. Yang, Z. Li, and Z. Liang, "An encoder-decoder framework translating natural language to database queries," in *Proceedings of the 27th International Joint Conference on Artificial Intelligence*, ser. IJCAI'18. AAAI Press, 2018, p. 3977–3983. [Online]. Available: https://doi.org/10.24963/ijcai.2018/553

[29] H. Liu, Y. Shi, J. Zhang, X. Wang, H. Li, and F. Kong, "Multi-hop relational graph attention network for text-to-sql parsing," in *2023 International Joint Conference on Neural Networks (IJCNN)*, 2023, pp. 1–8.

[30] B. Hui, R. Geng, L. Wang, B. Qin, Y. Li, B. Li, J. Sun, and Y. Li, "S^2SQL: Injecting syntax to question-schema interaction graph encoder for text-to-SQL parsers," in *Findings of the Association for Computational Linguistics: ACL 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 1254–1262. [Online]. Available: https://aclanthology.org/2022.findings-acl.99

[31] J. Li, B. Hui, R. Cheng, B. Qin, C. Ma, N. Huo, F. Huang, W. Du, L. Si, and Y. Li, "Graphix-t5: mixing pre-trained transformers with graph-aware layers for text-to-sql parsing," in *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i11.26536

[32] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds. Online: Association for Computational Linguistics, Jul. 2020, pp. 7567–7578. [Online]. Available: https://aclanthology.org/2020.acl-main.677

[33] K. Xu, L. Wu, Z. Wang, Y. Feng, and V. Sheinin, "SQL-to-text generation with graph-to-sequence model," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds. Brussels, Belgium: Association for Computational Linguistics, Oct.-Nov. 2018, pp. 931–936. [Online]. Available: https://aclanthology.org/D18-1112

[34] Y. Zheng, H. Wang, B. Dong, X. Wang, and C. Li, "HIE-SQL: History information enhanced network for context-dependent text-to-SQL semantic parsing," in *Findings of the Association for Computational Linguistics: ACL 2022*, S. Muresan, P. Nakov, and A. Villavicencio, Eds. Dublin, Ireland: Association for Computational Linguistics, May 2022, pp. 2997–3007. [Online]. Available: https://aclanthology.org/2022.findings-acl.236

[35] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, and D. Zhang, "Towards complex text-to-SQL in cross-domain database with intermediate representation," in *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, A. Korhonen, D. Traum, and L. Màrquez, Eds. Florence, Italy: Association for Computational Linguistics, Jul. 2019, pp. 4524–4535. [Online]. Available: https://aclanthology.org/P19-1444

[36] H. Li, J. Zhang, C. Li, and H. Chen, "Resdsql: decoupling schema linking and skeleton parsing for text-to-sql," in *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, ser. AAAI'23/IAAI'23/EAAI'23. AAAI Press, 2023. [Online]. Available: https://doi.org/10.1609/aaai.v37i11.26535

[37] T. Scholak, N. Schucher, and D. Bahdanau, "PICARD: Parsing incrementally for constrained auto-regressive decoding from language models," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, M.-F. Moens, X. Huang, L. Specia, and S. W.-t. Yih, Eds. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 9895–9901. [Online]. Available: https://aclanthology.org/2021.emnlp-main.779

[38] L. Wang, B. Qin, B. Hui, B. Li, M. Yang, B. Wang, B. Li, J. Sun, F. Huang, L. Si, and Y. Li, "Proton: Probing schema linking information from pre-trained language models for text-to-sql parsing," in *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, ser. KDD '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 1889–1898. [Online]. Available: https://doi.org/10.1145/3534678.3539305

[39] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *Proc. VLDB Endow.*, vol. 17, no. 5, p. 1132–1145, may 2024. [Online]. Available: https://doi.org/10.14778/3641204.3641221

[40] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," in *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine, Eds., vol. 36. Curran Associates, Inc., 2023, pp. 36 339–36 348. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf

[41] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, lu Chen, J. Lin, and D. Lou, "C3: Zero-shot text-to-sql with chatgpt," 2023. [Online]. Available: https://arxiv.org/abs/2307.07306

[42] H. Li, J. Zhang, H. Liu, J. Fan, X. Zhang, J. Zhu, R. Wei, H. Pan, C. Li, and H. Chen, "Codes: Towards building open-source language models for text-to-sql," *Proc. ACM Manag. Data*, vol. 2, no. 3, may 2024. [Online]. Available: https://doi.org/10.1145/3654930

[43] L. Nan, Y. Zhao, W. Zou, N. Ri, J. Tae, E. Zhang, A. Cohan, and D. Radev, "Enhancing text-to-SQL capabilities of large language models: A study on prompt design strategies," in *Findings of the Association for Computational Linguistics: EMNLP 2023*, H. Bouamor, J. Pino, and K. Bali, Eds. Singapore: Association for Computational Linguistics, Dec. 2023, pp. 14 935–14 956. [Online]. Available: https://aclanthology.org/2023.findings-emnlp.996

[44] T. Ren, Y. Fan, Z. He, R. Huang, J. Dai, C. Huang, Y. Jing, K. Zhang, Y. Yang, and X. S. Wang, "Purple: Making a large language model a better sql writer," 2024. [Online]. Available: https://arxiv.org/abs/2403.20014

[45] Z. Gu, J. Fan, N. Tang, L. Cao, B. Jia, S. Madden, and X. Du, "Few-shot text-to-sql translation using structure and content prompt learning," *Proc. ACM Manag. Data*, vol. 1, no. 2, jun 2023. [Online]. Available: https://doi.org/10.1145/3589292

[46] P. Ni, R. Okhrati, S. Guan, and V. Chang, "Knowledge graph and deep learning-based text-to-graphql model for intelligent medical consultation chatbot," *Information Systems Frontiers*, vol. 26, no. 1, p. 137–156, jul 2022. [Online]. Available: https://doi.org/10.1007/s10796-022-10295-0

[47] M. Staniek, R. Schumann, M. Züfle, and S. Riezler, "Text-to-OverpassQL: A natural language interface for complex geodata querying of OpenStreetMap," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 562–575, 2024. [Online]. Available: https://aclanthology.org/2024.tacl-1.31

[48] Y. Song, R. C.-W. Wong, and X. Zhao, "Speech-to-sql: toward speech-driven sql query generation from natural language question," *The VLDB Journal*, vol. 33, pp. 1179–1201, 2024. [Online]. Available: https://doi.org/10.1007/s00778-024-00837-0

[49] Y. Song, R. C.-W. Wong, X. Zhao, and D. Jiang, "Voicequerysystem: A voice-driven database querying system using natural language questions," in *Proceedings of the 2022 International Conference on Management of Data*. Association for Computing Machinery, 2022, pp. 2385–2388. [Online]. Available: https://doi.org/10.1145/3514221.3520158

[50] F. Özcan, A. Quamar, J. Sen, C. Lei, and V. Efthymiou, "State of the art and open challenges in natural language interfaces to data," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. Association for Computing Machinery, 2020, pp. 2629–2636. [Online]. Available: https://doi.org/10.1145/3318464.3383128

[51] J. Lu, Y. Song, H. Zhang, C. J. Zhang, K. Wu, and R. C.-W. Wong, "Towards robustness of text-to-visualization translation against lexical and phrasal variability," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, 2025, pp. 793–806.

[52] Y. Song, J. Lu, X. Zhao, R. C.-W. Wong, and H. Zhang, "Demonstration of fevisqa: Free-form question answering over data visualization," in *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, 2024, pp. 5417–5420.

[53] Y. Song, J. Lu, Y. Song, C. C. Cao, R. C.-W. Wong, and H. Zhang, "Fevisqa: Free-form question answering over data visualizations," in *2025 IEEE 41st International Conference on Data Engineering (ICDE)*, 2025, pp. 2726–2739.

[54] Y. Song, J. Lu, and R. C.-W. Wong, "Covis: Neural and llm-driven multi-turn interactions for conversational text-to-visualization generation," *The VLDB Journal*, vol. 35, no. 1, Nov. 2025. [Online]. Available: https://doi.org/10.1007/s00778-025-00954-4

[55] J. Lu, Y. Song, C. Zhang, and R. C.-W. Wong, "Multivis-agent: A multi-agent framework with logic rules for reliable and comprehensive

cross-modal data visualization," *Proc. ACM Manag. Data*, vol. 4, no. 1, Apr. 2026. [Online]. Available: https://doi.org/10.1145/3786670

[56] J. Lu, J. Lu, C. Zhang, Y. Song, and R. C.-W. Wong, "Demonstration of multivis-agent: Interactively generating reliable visualizations via logic-rule agents," in *Companion of the International Conference on Management of Data*, ser. SIGMOD Companion '26. New York, NY, USA: Association for Computing Machinery, 2026, pp. 98–101. [Online]. Available: https://doi.org/10.1145/3788853.3801584