# DATABASE CONTEXT COMPRESSION FOR TEXT-TO-SQL ON REAL-WORLD LARGE DATABASES

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

Recent progress on Text-to-SQL has been driven by stronger language models and richer prompting strategies, yet performance on real enterprise benchmarks such as Spider 2.0 and BIRD remains far below that on classical academic datasets. We argue that a central bottleneck on such benchmarks is the way the database is presented to the model. Real databases contain wide tables with repeated audit columns, large families of homogeneous partitioned tables, opaque machine-generated identifiers whose meaning lives only in column descriptions, and long data dictionaries in which only a small, query-dependent fraction is actually relevant. Existing query-aware approaches—schema linking, broadly construed to include retrieval-style schema subsetting—attempt to filter this raw context, but often operate on a representation that is structurally redundant, semantically verbose and documentation-heavy. We re-frame the problem as one of *database context compression*: a database-side rewrite that increases the information density of schema structure and semantic descriptors, complemented by query-aware purification of external documents. We introduce the SGCF (Support–Gain Component Factorization) principle as a common accounting abstraction for repeated column groups, isomorphic table templates and shared semantic-tag components; question-relevant evidence purification addresses the distinct, query-dependent document layer. Building on this abstraction, we present DBCC, a two-phase database-side middleware that performs query-agnostic structural and semantic re-encoding offline, and lightweight query-aware evidence purification online. DBCC is model-agnostic and pipeline-agnostic, and can be inserted before schema linking or generation in existing Text-to-SQL systems. On Spider 2.0-Snow and BIRD, DBCC reduces input tokens by up to $75\times$ (on the most challenging LARGE bucket of Spider 2.0-Snow, $2.6\,\text{M} \rightarrow 34.7\text{K}$ tokens). On that bucket, where the raw input is over budget and produces no completed predictions, the compressed input reaches $56.5\%$ strict recall under DeepSeek-V3.2 and $63.1\%$ under Claude-Opus-4.7. DBCC also yields a 1.8–1.9 percentage-point end-to-end EX improvement when stacked on top of each of three recent Text-to-SQL systems. Code and evaluation artifacts are available in the anonymous repository at https://anonymous.4open.science/r/SchemaCompression/.

## 1 INTRODUCTION

Text-to-SQL translates natural-language questions into executable SQL and is a foundational interface for making organizational data accessible to non-expert users (Li & Jagadish, 2014; Yaghmazadeh et al., 2017; Yu et al., 2018; Saha et al., 2016; Baik et al., 2019; 2020). Large language models (LLMs) have driven rapid progress on classical academic benchmarks (Wang et al., 2020; Li et al., 2023a; Pourreza & Rafiei, 2023; Talaei et al., 2024; Pourreza et al., 2024; Gao et al., 2024), yet the same systems remain brittle on enterprise databases. Such databases may expose thousands of columns, opaque business identifiers, semi-structured fields, and extensive documentation. Reliable access to this setting is therefore essential for moving Text-to-SQL from curated benchmarks to real data-intensive applications.

Recent work addresses this challenge primarily through query-aware schema linking (Wang et al., 2020; Li et al., 2023a; Lin et al., 2020; Cao et al., 2024), retrieval-style schema subsetting (Kothyari



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

et al., 2023; Wang et al., 2025b; Li et al., 2024a; Ma et al., 2025), and iterative agentic exploration (Talaei et al., 2024; Deng et al., 2025; Wang et al., 2025a; 2026). These methods differ in how they select or reason over context, but they share a consequential assumption: the raw database context is an adequate representation of the underlying information, so the central problem is to identify the relevant portion for each question. As enterprise schemas are shaped by denormalization, audit conventions, physical partitioning, machine-generated names, and independently authored dictionaries, however, physical context size can grow much faster than its task-relevant information content.

**Key Insight of Database Context for Text-to-SQL.** A real database is not merely a larger collection of independent schema elements. Its representation contains repeated structures across tables, recurring semantics across column descriptions, and long documents whose usefulness depends on the question. Applying query-aware selection directly to this representation implicitly assumes that selection can recover the relevant evidence before redundancy exhausts the context budget or obscures that evidence. This raises two critical questions: **Does database scale itself cause the observed failure, or does the low information density of its representation?** and **Can query-aware filtering alone overcome a representation dominated by repeated and weakly focused context?**

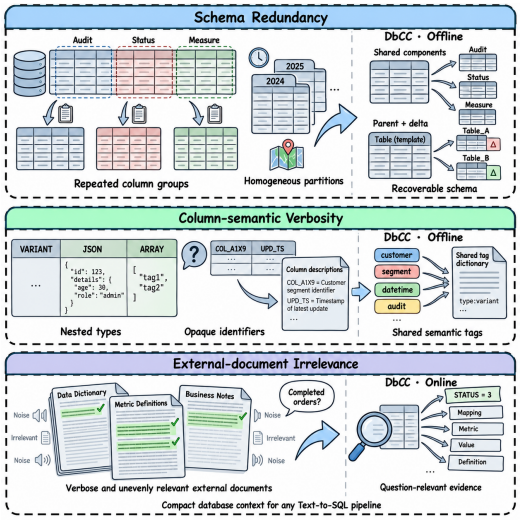



Figure 1: Database context compression for enterprise Text-to-SQL. The left side illustrates a raw database context containing wide tables, repeated partition schemas, opaque column identifiers, and lengthy documentation. The right side shows how DBCC rewrites these inputs into the compact view $D'(q) = (S', M', E'(q))$, which can be consumed directly by existing schema-linking and SQL-generation pipelines.

To answer these questions, we conduct three exploratory studies in Section 2. First, raw input grows from 9.6K tokens on small databases to 2.6M on large databases, while strict schema recall falls from 61.1% to 0.0%. Second, our context audit attributes the majority of this growth to repeated structural and semantic material rather than new task content. Third, under a matched token budget, selecting from raw schema items remains substantially worse than first collapsing representation-level redundancy. These results indicate that existing methods inherit a common failure ceiling from their input: *large-database Text-to-SQL is constrained not only by what context is selected, but by how the database is represented before selection begins*.

Motivated by these observations, we propose DBCC, a database-side middleware for *database context compression*. DBCC uses the SGCF (Support–Gain Component Factorization) principle to represent information shared by many carriers once as a reusable component. Its offline phase performs exactly reversible structural factorization and componentization over normalized semantic tags once per database, while its lightweight online phase purifies question-dependent documents into schema-anchored evidence. The resulting high-density view can be consumed by existing schema linkers and SQL generators without changing their internal reasoning procedures.

Our contributions are summarized as follows:

1. **Conceptually**, we expose an under-examined raw-context assumption in existing large-database Text-to-SQL systems. Through three exploratory studies, we show that physical schema growth contains substantial representational redundancy and that query-aware filtering alone does not fully overcome the resulting information-density bottleneck in our evaluation.
2. **Technically**, we formulate database context compression and introduce DBCC. Its SGCF abstraction supports exact structural factorization and componentization of normalized semantic tags, while a separate query-aware operator purifies external evidence.
3. **Experimentally**, we evaluate DBCC across database scales, LLMs, benchmarks, and downstream systems. On large Spider 2.0-Snow databases, it reduces input from 2.6M to 34.7K tokens

and raises strict schema recall from $0.0\%$ to $56.5\%$ under DeepSeek-V3.2, while consistently improving end-to-end execution accuracy when integrated with three recent Text-to-SQL systems.

## 2 Observation and Motivation

We begin from the assumption shared by most existing systems: that a large database is simply a larger candidate set over which a sufficiently capable query-aware method should search. We test this assumption through three controlled diagnostic experiments.

### 2.1 Performance Collapse under Raw-Context Scaling

We first run the same DeepSeek-V3.2 schema-linking and SQL-generation pipeline on Spider 2.0-Snow databases grouped by physical column count. Table 1 shows that growth is not gradual: the raw context reaches 2.6M tokens in the LARGE bucket, beyond the usable input budget, and strict recall (SRR) and execution accuracy (EX) both fall to zero. Even the MEDIUM bucket exhibits a sharp utility drop despite fitting nominal context windows. This pattern suggests that the failure begins well before hard overflow: as the context expands, relevant schema evidence becomes increasingly difficult to identify and exploit amid redundant and weakly discriminative content.

Table 1: Scaling diagnostic on uncompressed Spider 2.0-Snow contexts. The measurements correspond to the raw baseline in Table 4.

| Bucket | Columns | Input tokens | SRR | EX |
|---|---|---|---|---|
| SMALL | $< 1{,}000$ | 9.6K | 61.1 | 28.6 |
| MEDIUM | 1,000–10,000 | 166.0K | 44.8 | 9.4 |
| LARGE | $\geq 10{,}000$ | 2.6M | 0.0 | 0.0 |

***Observation I:*** *Text-to-SQL utility deteriorates before the raw database context reaches a hard context-window limit and collapses once enterprise-scale schemas exceed the usable budget; nominal context capacity alone does not guarantee usable evidence.*

### 2.2 Anatomy of Enterprise Database Context

Treating large databases as a quantitative scaling problem misses what makes them qualitatively different. Inspecting the databases shipped with Spider 2.0 (Lei et al., 2024) and BIRD (Li et al., 2023b) reveals that the growth of physical schema size is dominated by *engineering artefacts* rather than business semantics:

- **Wide tables and repeated column groups.** For performance and historical reasons, business tables are often denormalized, and identical groups of audit, status or measurement columns are pasted across many tables.
- **Homogeneous partitioned tables.** Time, region or version sharding produces large families of physical tables whose schemas are essentially identical to a common parent.
- **Composite and semi-structured columns.** `VARIANT`, `JSON`, `ARRAY` and `STRUCT` columns hide the discriminative information inside nested attributes, where neither column name nor type alone is sufficient for matching.
- **Opaque machine-generated identifiers.** Many column names are abbreviations, codes or auto-generated tokens whose business meaning lives only in external column descriptions.
- **Verbose and unevenly relevant external documents.** Data dictionaries, metric definitions and business notes provide essential disambiguation, but each document is long, only a small fraction is relevant to any given question, and the rest acts as in-context noise.

These five observations are not five independent problems. They cluster into three challenges, each defined by the layer of the database context at which low information density arises and by whether that density is intrinsic to the database or to the question.

**Challenge #1 (Schema redundancy).** Observations 1–2 inflate the *schema* layer with content that is repeated across tables: identical groups of audit, status or measurement columns reappear in many wide tables, and isomorphic schemas reappear across families of sharded tables. The redundancy is structural and query-agnostic—a column group repeated in 30 tables is repeated regardless of who is asking.

**Challenge #2 (Column-semantic verbosity).** Observations 3–4 inflate the *column-semantics* layer: composite and semi-structured columns force the system to inspect nested attributes that neither column name nor type alone disambiguates, while opaque machine-generated identifiers shift the burden of meaning onto verbose external column descriptions. The cost paid per column is high, and—because similar concepts are described differently across tables—the same semantic content is paid for many times.

**Challenge #3 (External-document irrelevance).** Observation 5 inflates the *external-knowledge* layer in a qualitatively different way: each individual document is internally non-redundant, but only a small, question-dependent fraction of any document is relevant to a given question; the remainder acts as in-context noise. Unlike Challenges #1–#2, the redundancy here cannot be removed once and for all—it depends on the question.

The combined effect is that real database context is simultaneously *too large to fit* and *too unfocused to use*. To quantify its composition, we analyze the Spider 2.0-Snow databases in each scale bucket and assign every serialized token to unique schema content, repeated structural or semantic content, question-irrelevant external documentation, or unique task content. Table 2 reports the measured composition. Repeated context grows from $38.4\%$ in SMALL databases to $92.1\%$ in LARGE databases, while unique task content falls from $48.9\%$ to $3.6\%$.

The important trend is not simply that large databases contain more tokens, but that the marginal tokens carry progressively less new information. This explains why nominally long context windows do not solve the problem: they enlarge the container without changing the density of discriminative evidence.

Table 2: Context-composition audit. "Repeated" is repeated structural or semantic material; "irrelevant docs" is documentation judged irrelevant to the sampled question.

| Bucket | Repeated context | Irrelevant docs | Unique task content |
|---|---|---|---|
| SMALL | 38.4% | 12.7% | 48.9% |
| MEDIUM | 71.6% | 10.8% | 17.6% |
| LARGE | 92.1% | 4.3% | 3.6% |

***Observation II:*** *As databases grow, repeated structural and semantic material expands substantially faster than unique task content, making enterprise database scale primarily an information-density problem rather than a uniform candidate-set expansion.*

### 2.3 REPRESENTATION QUALITY UNDER A MATCHED BUDGET

Finally, we distinguish representation quality from token budget. On LARGE databases, we compare three inputs capped at 32K tokens: prefix truncation of the raw serialization, a query-aware BM25 subset over raw schema items, and an oracle redundancy-collapsed representation constructed using gold knowledge of repeated groups but without our proposed algorithms. Table 3 reports the measured results under this controlled budget. Query-aware selection substantially improves over prefix truncation, but oracle redundancy collapse provides a further $22.8$-point SRR and $3.8$-point EX gain, showing that selection over a low-density representation leaves substantial utility unrealized.

Table 3: Matched-budget diagnostic on the LARGE bucket. The input budget is 32K tokens for every row.

| Representation | SRR | NSR | EX |
|---|---|---|---|
| Raw prefix truncation | 6.8 | 12.4 | 1.7 |
| Query-aware subset of raw items | 31.9 | 49.8 | 7.2 |
| Oracle redundancy collapse | 54.7 | 74.1 | 11.0 |

***Observation III:*** *Under the same input budget, selecting elements from the raw database representation remains markedly worse than first collapsing representation-level redundancy; representation quality and query-aware relevance are complementary rather than interchangeable.*

**Summary.** Building on the above observations, our goal is to increase the information density of the database context before downstream schema linking and SQL generation. DBCC applies query-agnostic structural factorization to repeated column groups and table families (addressing ***Observation II***), shared semantic components to recurring column meanings (also addressing ***Observation II***), and question-aware evidence purification only where relevance is intrinsically query-dependent (addressing ***Observation III***). The resulting offline/online split keeps large contexts within a usable budget while preserving recoverable schema content (addressing ***Observation I***), directly motivating the formulation and architecture in Section 3.1.

# 3 METHODOLOGY

## 3.1 PROBLEM FORMULATION

A database context contains the schema $S$, column-level semantics $M$, and external knowledge $E$, denoted by $D = (S, M, E)$. Given a question $q$, a downstream Text-to-SQL system $\Pi$ predicts $\hat{y} = \Pi(q, D)$. We seek a compressor

$$\mathcal{C} : D \mapsto D'(q) = (S', M', E'(q))$$

that reduces input cost while preserving or improving downstream utility:

$$\mathrm{Cost}(D') \ll \mathrm{Cost}(D), \qquad \mathrm{Util}(\Pi, D', q) \geq \mathrm{Util}(\Pi, D, q). \tag{1}$$

$S'$ is an exactly reversible, query-agnostic structural view cached per database. $M'$ is reversible with respect to the normalized tag representation produced by our semantic extractor, although the preceding conversion from free-form descriptions to tags is lossy. Only $E'(q)$ is computed online and, by construction, it is a query-conditioned extraction rather than a lossless encoding of the source documents. Appendix B states these guarantees separately.

To express the reusable part of this rewrite, we model an information layer as $\langle \mathcal{O}, U, \kappa \rangle$: $\mathcal{O}$ is a set of carrier objects, $U(o) \subseteq \mathcal{V}$ is the set of information units carried by $o$, and $\kappa$ measures serialization cost. A component $X \subseteq \mathcal{V}$ is reusable when it occurs in multiple carriers. Its support and width are

$$s(X) = \big|\{o \in \mathcal{O} : X \subseteq U(o)\}\big|, \qquad w(X) = |X|.$$

This layer abstraction covers tables carrying column signatures and columns carrying normalized semantic tags. It deliberately excludes external documents: whether a document span is useful depends on $q$, so that layer is handled by online evidence extraction rather than an offline lossless factorization.

## 3.2 THE DBCC FRAMEWORK

DBCC is based on SGCF (Support–Gain Component Factorization). For a candidate information component $X$, let $s(X)$ be the number of carriers containing it and $w(X) = |X|$ its width. Materializing $X$ once and replacing its occurrences with references yields the approximate saving

$$\mathrm{gain}(X) = s(X)w(X) - \big(s(X) + w(X)\big). \tag{2}$$

For an admissible component family $\mathcal{X}$, SGCF uses the coverage-style objective

$$\max_{\mathcal{X}} \sum_{X \in \mathcal{X}} \mathrm{gain}(X) \quad \text{s.t. layer-specific support, overlap and readability constraints.} \tag{3}$$

The reward $s(X)w(X)$ measures repeated material removed, whereas $s(X) + w(X)$ pays once for the component definition and once per reference. The structural and semantic layers greedily select admissible positive-gain components, using tables for structural groups, table families for templates, and columns for semantic tags. This unit-cost objective is a selection proxy rather than an exact tokenizer-level optimum; all reported costs are measured on the final serialization. External documents require a different treatment because their relevance is question-dependent and are therefore purified online. Appendix B gives the full scope and assumptions, and Appendix D provides the pseudocode.

Figure 2 shows the two phases. Phase I rewrites structure and column semantics once per database and caches $(S', M')$. Phase II purifies external documents only after $q$ arrives, producing $E'(q)$. The final view can replace the raw context in an existing linker or generator without changing its model or decoding logic.

## 3.3 STRUCTURAL COMPRESSION

DBCC captures two complementary forms of schema redundancy. **Column-group factorization** (Sec. 3.3.1) identifies column signatures that co-occur across many tables and stores each accepted group once as a shared component. **Template hierarchy** (Sec. 3.3.2) clusters isomorphic physical tables and represents a derived family by a parent template plus its column delta. The first removes horizontal repetition across wide tables; the second removes vertical repetition across partitioned families. Both are reversible, so any referenced group or template can be expanded before downstream execution.

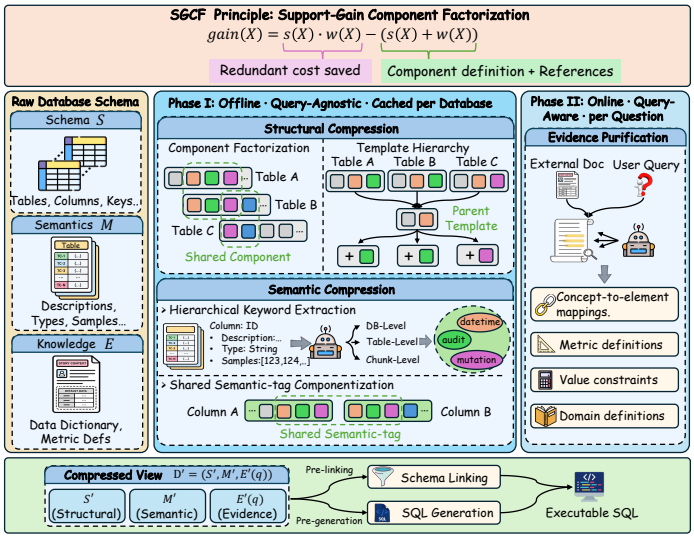



Figure 2: Two-phase DBCC. The offline phase caches structural components $S'$ and normalized semantic components $M'$; the online phase extracts schema-anchored evidence $E'(q)$. Their composition $D'(q)$ is consumed by downstream Text-to-SQL pipelines.

#### 3.3.1 COMPONENT FACTORIZATION FOR WIDE TABLES

Let $\mathcal{T}$ be the table set and let $\phi(c) = \{T \in \mathcal{T} : c \in T\}$ be the support set of column signature $c$. Instead of enumerating exponentially many column subsets, we form one candidate

$$X_A = \{c : \phi(c) = A\}$$

for each distinct support set $A$. A candidate is admissible when $s(X_A) \geq \tau_s$, $w(X_A) \geq \tau_w$, and $\text{gain}(X_A) \geq \tau_g$. Candidates are processed by decreasing gain, subject to disjointness and a per-table reference quota $K$. The restriction to exact support classes makes construction tractable, but is heuristic: a useful group whose columns have different, overlapping supports may be missed. Accepted groups are stored once and replaced by references, so the original table signatures remain exactly recoverable. Appendix B analyzes this restriction and overlap handling; Algorithm 1 gives the complete procedure.

#### 3.3.2 TEMPLATE HIERARCHY FOR HOMOGENEOUS PARTITIONS

Let $B_i$ group physical tables with the same signature $\text{Sig}(B_i)$. For each non-root family, DBCC chooses the most specific admissible parent

$$\text{Par}(B_i) = \arg \max_{B_j : \text{Sig}(B_j) \subseteq \text{Sig}(B_i)} |\text{Sig}(B_j)|, \qquad \Delta_i = \text{Sig}(B_i) \setminus \text{Sig}(\text{Par}(B_i)).$$

The link is retained only when its serialization gain is positive and $|\Delta_i|/|\text{Sig}(B_i)| \leq \tau_r$; otherwise $B_i$ remains a root. Thus, an isomorphic family is represented by stable physical-table identifiers, one shared signature, and small deltas along accepted inheritance edges. Column-group factorization removes repeated subsets across tables, whereas this hierarchy removes repetition of entire or near-entire signatures. Parent selection, identity preservation and the full construction appear in Appendix B and Algorithm 2.

### 3.4 SEMANTIC COMPRESSION

For each column $c_i$ with description $d_i$, type $t_i$, and samples $v_i$, DBCC extracts a normalized semantic tag

$$K_i = \text{Norm}\big(K_{\text{desc}}(d_i, v_i) \cup K_{\text{type}}(t_i)\big).$$

Extraction proceeds from database to table and chunk level so that synonyms across tables share a vocabulary. Repeated subsets of $K_i$ are then factorized with Eq. equation 2; each column retains

references to shared tag components plus unmatched residual tags. Appendix B describes hierarchical propagation, candidate refresh, and the distinction from structural components; Algorithm 3 gives the selection procedure. Formally, after selecting semantic components $\mathcal{C}_M$, column $c_i$ is represented as

$$\widehat{K}_i = (R_i, U_i), \qquad R_i = \{X \in \mathcal{C}_M : X \subseteq K_i\}, \quad U_i = K_i \setminus \bigcup_{X \in R_i} X.$$

Candidates up to width $w_{\max}$ are ordered by support, with gain and width as tie-breakers; after accepting a component, affected residual sets and candidate statistics are refreshed. Disjoint tag components make decoding deterministic, so $(R_i, U_i)$ recovers the normalized $K_i$ exactly, although the earlier description-to-tag normalization is not reversible. Appendix B details hierarchical vocabulary propagation and Algorithm 3 gives the selection procedure.

### 3.5 External-Knowledge Compression

External documents are question-dependent rather than globally redundant. DBCC therefore computes $E'(q) = \mathrm{Purify}(q, E)$ online: an LLM extracts a small set of typed atoms and a deterministic pass anchors them to schema elements. The atoms cover concept-to-element mappings, metric definitions, encoded filter values, and domain definitions. Unlike passage retrieval, the output is directly consumable by schema linking and SQL generation. Prompt schemas, validation, fallback behavior, and examples are deferred to Appendix B.

### 3.6 Representation Contract and Reversibility

Appendix B specifies the component registry, stable identifiers, exact recovery guarantee for structural signatures and normalized tags, source-level fallback for lossy textual transformations, and the serialization boundary at which token cost is measured.

### 3.7 Integration with Text-to-SQL Pipelines

Appendix B details how the single interface $D \mapsto D'(q)$ is inserted before schema linking or SQL generation without retraining, together with the minimal downstream assumptions and offline-cost amortization model.

## 4 Experiments

We design our study around five questions: (Q1) **How much** can DBCC compress real database context, and **at what utility cost or gain** for downstream schema linking? (Q2) **How do the three operators interact**, and when are their effects complementary or antagonistic? (Q3) **Does** DBCC **transfer** across different LLMs, different benchmarks, and different downstream pipelines, and is the offline phase **economically amortizable**? (Q4) **What does the cost–utility trade-off look like**—is there a sweet spot, and is the curve monotone? (Q5) **Where does** DBCC **still fail**, and which residual errors should the next iteration target? The complete experimental setup is provided in Appendix C.

### 4.1 Ablation and Operator Interactions

We now evaluate the three operators jointly with a clean $2^3$-style ablation, where each operator is independently turned on or off. Structural compression uses template hierarchy, semantic uses frequency-priority, and external knowledge uses purified evidence; *nothing else changes between rows*, so the differences are attributable to the operators themselves. Table 4 summarizes all eight configurations on the three buckets.

**Cumulative gains** On Medium databases, full DBCC improves SRR by 11.5 percentage points and NSR by 18.6 percentage points over the uncompressed baseline, while reducing tokens by 91.4% ↑ (166.0K → 14.3K). On Large databases, the raw serialization exceeds the usable input budget and is therefore reported as an overflow case with zero completed predictions; DBCC makes the setting executable and obtains 56.5% SRR together with a 98.7% ↑ token reduction. This comparison primarily demonstrates feasibility at that scale and should not be interpreted as a controlled 56.5-point

Table 4: Fine-grained ablation of DBCC on Spider 2.0-Snow. LLM: DeepSeek-V3.2. Columns S/Sem/EK indicate whether structural, semantic and external-knowledge operators are on. The row (✗,✗,✗) is the uncompressed baseline; (✓,✓,✓) is full DBCC. Best per metric per bucket in **bold**, second-best underlined.

| | | | SMALL | | | | | MEDIUM | | | | | LARGE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| S | Sem | EK | SRR | NSR | NSP | Tok | EX | SRR | NSR | NSP | Tok | EX | SRR | NSR | NSP | Tok | EX |
| ✗ | ✗ | ✗ | **61.1** | 84.3 | **78.6** | 9.6K | 28.6 | 44.8 | 67.3 | 57.1 | 166.0K | 9.4 | 0.0 | 2.9 | 50.0 | 2.6M | 0.0 |
| ✗ | ✗ | ✓ | **62.3** | 84.3 | 76.1 | 9.4K | 29.0 | 45.8 | 77.8 | **62.2** | 165.4K | 9.9 | 4.3 | 4.3 | 60.0 | 2.6M | 1.0 |
| ✗ | ✓ | ✗ | 60.6 | **85.0** | 74.8 | 5.1K | 29.1 | 49.0 | 76.4 | 47.5 | 43.6K | 14.2 | 8.7 | 13.3 | 60.9 | 712.4K | 2.6 |
| ✓ | ✗ | ✗ | 60.6 | 83.8 | 75.5 | 7.5K | 30.4 | 50.0 | 81.4 | 57.4 | 29.9K | **17.2** | 47.8 | 69.5 | 71.6 | 100.7K | **11.6** |
| ✗ | ✓ | ✓ | 60.0 | 82.0 | 75.3 | 5.0K | 29.3 | 47.9 | 74.2 | 48.7 | 43.1K | 12.7 | 13.0 | 24.3 | 71.8 | 716.3K | 3.4 |
| ✓ | ✗ | ✓ | 59.1 | 83.0 | 74.7 | 7.4K | 30.3 | 51.0 | 78.9 | 54.6 | 29.3K | 14.6 | 43.5 | 75.7 | 69.1 | 100.6K | 9.4 |
| ✓ | ✓ | ✗ | **61.1** | 83.3 | 71.4 | 4.7K | 30.7 | 55.2 | 77.3 | 58.1 | 14.9K | 16.4 | 52.2 | 76.7 | 72.9 | 34.9K | 11.0 |
| ✓ | ✓ | ✓ | 61.7 | 82.4 | 71.8 | **4.5K** | **31.2** | **56.3** | **85.9** | 55.7 | **14.3K** | **17.2** | **56.5** | **78.6** | **73.3** | **34.7K** | **11.6** |

quality gain over a runnable raw-input system. On SMALL databases the utility is already saturated by the baseline; here DBCC preserves SRR/NSR within one to two percentage points while still cutting tokens by 53.1% ↑.

**Operator interactions** The factorial table shows that disjoint representation layers do not imply independent downstream effects. On the MEDIUM bucket, structural and semantic compression together improve SRR more than either operator alone, and adding evidence purification to the structural configuration yields a further gain. By contrast, semantic compression and evidence purification without structural compression are mildly antagonistic, and on the LARGE bucket adding evidence purification to structural compression alone decreases SRR. These interactions are plausible because semantic tags and evidence atoms both depend on column-level alignment, even though they modify different fields of the representation. We therefore interpret the operators as *partly complementary*, not statistically orthogonal; structural compression is the main enabler on the largest schemas, while the other operators help in selected combined configurations.

### 4.2 END-TO-END INTEGRATION

To check whether a database-side rewrite translates to end-to-end gains, we plug DBCC into three recent Text-to-SQL systems—**AutoLink** (Wang et al., 2026), **ReFoRCE** (Deng et al., 2025) and **Apex-SQL** (Cao et al., 2026)—without modifying their pipelines, decoders, or post-processing. We restrict to Spider 2.0-Snow databases with >5,000 columns, where context organization most aggressively dominates accuracy.

Table 5 shows that on all three systems, simply replacing the raw context with the DBCC-compressed view improves EX (by 1.8% ↑, 1.8% ↑ and 1.9% ↑ respectively) while substantially reducing token consumption (57.0% ↑ for AutoLink, 73.5% ↑ for ReFoRCE and 59.6% ↑ for Apex-SQL). Apex-SQL is by a wide margin the strongest baseline on raw inputs (43.7% EX, more than twice that of AutoLink), reflecting the effectiveness of its agentic exploration loop at narrowing the database context per turn; even at this much higher operating point, DBCC compounds with that loop and adds a measurable EX gain at roughly 40% of the token cost. The absolute EX gains are modest (≤2%) and, as expected, smaller than the schema-linking improvements of Sec. 4.1: end-to-end EX is influenced by SQL planning, dialect handling, error recovery and post-processing, all of which are downstream of database context. DBCC only changes the input these systems consume, yet the EX improvement is positive on every one of the three systems,

Table 5: End-to-end plug-in integration with three recent Text-to-SQL systems on Spider 2.0-Snow databases with >5,000 columns. LLM: DeepSeek-V3.2. Best within each system in **bold**.

| System | DB context | EX | Tok |
|---|---|---|---|
| AutoLink | Raw | 19.6% | 61.0K |
| AutoLink | + DBCC | **21.4%** | **26.2K** |
| ReFoRCE | Raw | 16.1% | 16.6K |
| ReFoRCE | + DBCC | **17.9%** | **4.4K** |
| Apex-SQL | Raw | 43.7% | 48.3K |
| Apex-SQL | + DBCC | **45.6%** | **19.5K** |

suggesting that real Text-to-SQL stacks are bottlenecked by context quality even when they invest in agentic flows or self-refinement.

### 4.3 Comparison with Schema Linking and Pruning Baselines

We compare DBCC against three representative query-aware baselines: **Crush4SQL** (Kothyari et al., 2023), **LinkAlign** (Wang et al., 2025b) and **RSL-SQL** (Cao et al., 2024). We also report *stacked* settings for Crush4SQL and LinkAlign, where DBCC's rewritten schema is fed into the baseline as input, to test whether database-side rewriting and query-aware filtering are complementary in practice.

**Observations:** (i) DBCC outperforms the evaluated query-aware baselines on every reported metric while using $\sim 5\times$ fewer tokens, supporting our framing that database-side rewriting addresses a bottleneck not fully captured by query-side filtering. (ii) Stacking DBCC with Crush4SQL or LinkAlign yields a further gain of up to 2.7 SRR percentage points over standalone DBCC, suggesting practical complementarity; this result does not establish statistical independence between the method families. (iii) The stacked variants remain below 20K tokens, compared with 925K tokens for the raw serialization.

Table 6: Comparison with query-aware schema-linking baselines on Spider 2.0-Snow. LLM: GPT-4o. Best per metric in **bold** (higher is better for SRR/NSR/NSP/EX, lower for Tok).

| Method | SRR | NSR | NSP | Tok | EX |
|---|---|---|---|---|---|
| Raw | 38.2 | 56.6 | 55.7 | 925K | 14.6 |
| Crush4SQL | 47.5 | 67.4 | 58.1 | 102K | 17.9 |
| LinkAlign | 51.0 | 71.5 | 60.4 | 86K | 19.0 |
| RSL-SQL | 49.5 | 70.1 | 61.7 | 96K | 18.6 |
| DBCC | 60.9 | 81.0 | 67.4 | **18.0K** | 22.3 |
| DBCC + Crush4SQL | 62.0 | 82.4 | **68.9** | 19.4K | 22.9 |
| DBCC + LinkAlign | **63.6** | **83.5** | 68.5 | **17.6K** | **23.6** |

**Extended Investigations and Key Insights.** Beyond the core results above, we conduct extended investigations into how and when database context compression works. Appendix E separately evaluates the structural, semantic, and external-knowledge operators; verifies robustness across LLMs and benchmarks; traces the cost–utility Pareto frontier; measures offline-cost amortization; and categorizes residual failure modes. These analyses yield four consistent insights: structural compression is the principal enabler on very large schemas, semantic and evidence compression become increasingly important when business meaning resides outside identifiers, the gains persist across model and benchmark changes, and moderate compression improves both cost and utility whereas overly aggressive compression eventually removes salience. Full results are reported in Appendices E.1–E.8.

## 5 Limitations, Discussion and Conclusion

**Limitations and discussion.** DBCC has two practical limitations that motivate future work. First, the current offline phase rebuilds the compressed representation at database granularity after a schema or metadata update. Although this cost is amortized across queries, frequent updates may introduce avoidable maintenance overhead. Future work will develop incremental maintenance that detects affected tables, components and registry entries, and updates only the corresponding portion of the compressed view. Second, semantic-tag extraction and evidence purification rely on LLM-generated intermediate representations, so occasional normalization inconsistencies or imprecise schema anchors may reduce the salience of an otherwise recoverable element. Future work will incorporate calibrated extraction confidence, consistency checking and source-grounded verification, with selective expansion of the relevant component or fallback to the original description or passage when confidence is low.

**Conclusion.** This work identifies low-information-density database context as a central bottleneck in enterprise-scale Text-to-SQL and reframes context construction as a representation problem rather than only a query-time selection problem. We introduce DBCC, a two-phase framework that applies the SGCF accounting principle to repeated schema structures and normalized semantic components, together with a separate query-aware operator for external evidence. Across Spider 2.0-Snow and BIRD, DBCC substantially reduces input cost while improving the reported schema-linking and end-to-end results, and the observed gains transfer across the evaluated LLMs and downstream pipelines. More broadly, our results suggest that Text-to-SQL systems should consider database representation as a first-class component rather than treating the raw schema as a fixed interface.

### AI USE STATEMENT

We used generative AI tools solely for prose polishing and for assisting with related-work literature search. All other components of this work, including the DBCC formulation, the theoretical derivations, the figures, the code, and the experimental design, were completed by the human authors. We have reviewed all AI-assisted portions of the manuscript, and we take full responsibility for the final content of this work.

### ETHICS STATEMENT

This work evaluates on widely adopted public benchmarks and uses pretrained language models accessed under their respective research or service terms. It does not involve human subjects, private user data, or deployment in a safety-critical setting. The principal risks are those shared by LLM-based data interfaces, including erroneous SQL and unintended data access; practical deployments should therefore retain execution safeguards, access control, and human oversight.

### REPRODUCIBILITY STATEMENT

The source code and evaluation artifacts required to reproduce our experiments are available in the anonymous repository at https://anonymous.4open.science/r/SchemaCompression/. The appendix specifies the datasets, scale buckets, metrics, model configurations, default DBCC thresholds, algorithms, and evaluation protocols. The repository additionally provides the prompts, serialization templates, preprocessing scripts, and baseline configurations used to reproduce the reported compression, schema-linking, and end-to-end results.

## REFERENCES

C. Baik, H. V. Jagadish, and Y. Li. Bridging the semantic gap with SQL query logs in natural language interfaces to databases. In *Proc. IEEE Int. Conf. Data Engineering (ICDE)*, pp. 374–385, 2019.

C. Baik, Z. Jin, M. J. Cafarella, and H. V. Jagadish. Duoquest: A dual-specification system for expressive SQL queries. In *Proc. ACM SIGMOD Int. Conf. Management of Data*, pp. 2319–2329, 2020.

B. Cao, W. Liao, Y. Sun, D. Fang, H. Li, and W. Lam. APEX-SQL: Talking to the data via agentic exploration for Text-to-SQL. In *Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining (KDD)*, 2026.

R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, and K. Yu. LGESQL: Line graph enhanced text-to-SQL model with mixed local and non-local relations. In *Proc. ACL-IJCNLP*, pp. 2541–2555, 2021.

Z. Cao, Y. Zheng, Z. Fan, X. Zhang, W. Chen, and X. Bai. RSL-SQL: Robust schema linking in text-to-SQL generation. *arXiv preprint arXiv:2411.00073*, 2024.

M. Deng, A. Ramachandran, C. Xu, L. Hu, Z. Yao, A. Datta, and H. Zhang. ReFoRCE: A text-to-SQL agent with self-refinement, format restriction, and column exploration. In *ICLR Workshop on VerifAI*, 2025.

X. Deng, A. Hassan Awadallah, C. Meek, O. Polozov, H. Sun, and M. Richardson. Structure-grounded pretraining for text-to-SQL. In *Proc. NAACL*, pp. 1337–1350, 2021.

X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, J. Lin, D. Lou, et al. C3: Zero-shot text-to-SQL with ChatGPT. *arXiv preprint arXiv:2307.07306*, 2023.

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.*, 17(5):1132–1145, 2024.

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 *Proc. ACL*, pp. 4524–4535, 2019.

H. Jiang, Q. Wu, C.-Y. Lin, Y. Yang, and L. Qiu. LLMLingua: Compressing prompts for accelerated inference of large language models. In *Proc. EMNLP*, pp. 13358–13376, 2023.

M. Kothyari, D. Dhingra, S. Sarawagi, and S. Chakrabarti. CRUSH4SQL: Collective retrieval using schema hallucination for Text2SQL. In *Proc. EMNLP*, pp. 14054–14066, 2023.

F. Lei, J. Chen, Y. Ye, et al. Spider 2.0: Evaluating language models on real-world enterprise text-to-SQL workflows. *arXiv preprint arXiv:2411.07763*, 2024.

F. Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

H. Li, J. Zhang, C. Li, and H. Chen. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-SQL. *Proc. AAAI*, 37(11):13067–13075, 2023a.

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*, 2(3): 1–28, 2024a.

J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo, et al. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. *Adv. Neural Inf. Process. Syst.*, 36:42330–42357, 2023b.

X. Li, Q. Cai, Y. Shu, C. Guo, and B. Yang. AID-SQL: Adaptive in-context learning of text-to-SQL with difficulty-aware instruction and retrieval-augmented generation. In *Proc. IEEE Int. Conf. Data Engineering (ICDE)*, pp. 3945–3957, 2025.

Z. Li, X. Wang, J. Zhao, S. Yang, et al. PET-SQL: A prompt-enhanced two-stage text-to-SQL framework with cross-consistency. *arXiv preprint arXiv:2403.09732*, 2024b.

X. V. Lin, R. Socher, and C. Xiong. Bridging textual and tabular data for cross-domain text-to-SQL semantic parsing. In *Findings of EMNLP*, pp. 4870–4888, 2020.

A. Liu, X. Hu, L. Lin, and L. Wen. Semantic enhanced text-to-SQL parsing via iteratively learning schema linking graph. In *Proc. ACM SIGKDD Conf. Knowledge Discovery and Data Mining*, pp. 1021–1030, 2022.

N. F. Liu, K. Lin, J. Hewitt, A. Paranjape, M. Bevilacqua, F. Petroni, and P. Liang. Lost in the middle: How language models use long contexts. *Trans. Assoc. Comput. Linguist.*, 12:157–173, 2024.

H. Ma, Y. Shen, H. Liu, et al. DB-Explore: Automated database exploration and instruction synthesis for text-to-SQL. *arXiv preprint arXiv:2503.04959*, 2025.

M. Pourreza and D. Rafiei. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. *Adv. Neural Inf. Process. Syst.*, 36:36339–36348, 2023.

M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, et al. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. *arXiv preprint arXiv:2410.01943*, 2024.

N. Reimers and I. Gurevych. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proc. EMNLP-IJCNLP*, pp. 3982–3992, 2019.

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. In *Proc. IEEE Int. Conf. Data Engineering (ICDE)*, pp. 15–28, 2024.

S. Robertson and H. Zaragoza. *The Probabilistic Relevance Framework: BM25 and Beyond*, volume 4. Now Publishers Inc., 2009.

D. Saha, A. Floratou, K. Sankaranarayanan, U. F. Minhas, A. R. Mittal, and F. Özcan. ATHENA: An ontology-driven system for natural language querying over relational data stores. *Proc. VLDB Endow.*, 9(12):1209–1220, 2016.

T. Scholak, N. Schucher, and D. Bahdanau. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proc. EMNLP*, pp. 9895–9901, 2021.

S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi. CHESS: Contextual harnessing for efficient SQL synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

I. Trummer. Generating succinct descriptions of database schemata for cost-efficient prompting of large language models. *Proc. VLDB Endow.*, 17(11), 2024.

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 *Proc. ACL*, pp. 7567–7578, 2020.

B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, Q.-W. Zhang, Z. Yan, and Z. Li. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proc. COLING*, pp. 540–557, 2025a.

Y. Wang, P. Liu, and X. Yang. LinkAlign: Scalable schema linking for real-world large-scale multi-database text-to-SQL. In *Proc. EMNLP*, pp. 977–991, 2025b.

Z. Wang, Y. Zheng, Z. Cao, et al. AutoLink: Autonomous schema exploration and expansion for scalable schema linking in text-to-SQL at scale. In *Proc. AAAI Conf. Artificial Intelligence*, 2026. to appear.

N. Yaghmazadeh, Y. Wang, I. Dillig, and T. Dillig. SQLizer: Query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA):1–26, 2017.

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 *Proc. EMNLP*, pp. 3911–3921, 2018.

T. Yu, C.-S. Wu, X. V. Lin, B. Wang, Y. C. Tan, X. Yang, D. Radev, R. Socher, and C. Xiong. GraPPa: Grammar-augmented pre-training for table semantic parsing. In *Proc. ICLR*, 2021.

H. Zhang, R. Cao, L. Chen, H. Xu, and K. Yu. ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. *arXiv preprint arXiv:2310.17342*, 2023a.

Y. Zhang, A. Floratou, J. Cahoon, S. Krishnan, A. C. Müller, D. Banda, F. Psallidas, and J. M. Patel. Schema matching using pre-trained language models. In *Proc. IEEE Int. Conf. Data Engineering (ICDE)*, pp. 1558–1571, 2023b.

# A Related Work

## A.1 Text-to-SQL with LLMs

The transition from supervised parsers (Wang et al., 2020; Li et al., 2023a; Cao et al., 2021; Yu et al., 2021; Deng et al., 2021; Lin et al., 2020; Scholak et al., 2021; Guo et al., 2019) to LLM-based pipelines (Dong et al., 2023; Pourreza & Rafiei, 2023; Zhang et al., 2023a; Gao et al., 2024; Li et al., 2024b; Talaei et al., 2024; Pourreza et al., 2024; Ren et al., 2024; Li et al., 2025) has shifted attention from parser architecture toward task-context organization. Decomposition and multi-path methods such as DIN-SQL, CHESS, and CHASE-SQL improve reasoning over a supplied context, while multi-agent or self-refining systems such as MAC-SQL, REFORCE, and AUTOLINK repeatedly explore that context. These advances are complementary to DBCC: they modify how a model reasons, whereas we modify the database representation on which that reasoning begins.

## A.2 Schema linking and retrieval

Schema linking aligns question spans with database elements, either through fine-grained alignment inside a parser (Wang et al., 2020; Li et al., 2023a; Lin et al., 2020) or through a standalone linker or retriever (Cao et al., 2024; Wang et al., 2025b; Liu et al., 2022; Kothyari et al., 2023; Li et al., 2024a; Ma et al., 2025; Zhang et al., 2023b). These methods already address large candidate spaces and should not be viewed as assuming that every raw schema is directly usable. Their main operation is query-aware selection, whereas DBCC precomputes a reusable database-side representation before that selection. Its structural transformation preserves all table and column candidates exactly; semantic tags and purified evidence are compact, potentially lossy views with fallback pointers to their sources. The approaches are complementary in our stacked results for Crush4SQL and LinkAlign (Table 6).

### A.3 PROMPT COMPRESSION AND SCHEMA DESCRIPTION

Generic compressors such as LLMLINGUA (Jiang et al., 2023) remove token-level redundancy from arbitrary prompts, while the Lost-in-the-Middle analysis (Liu et al., 2024) explains why merely extending context can fail. SCHEMONIC (Trummer, 2024) is closest in spirit because it generates concise natural-language schema descriptions. DBCC instead exploits database-specific structural regularities—repeated column groups, table isomorphism, and shared semantic tags—to create reversible components rather than free-form summaries. Generic textual compression can still be applied to the residual prose after our rewrite.

## B DETAILED METHODOLOGY

This appendix provides the formal definitions, design rationale, implementation details, and integration analysis omitted from the main paper for space.

### B.1 PROBLEM FORMALIZATION

This section formalizes *database context compression* as a database-side rewrite problem.

#### B.1.1 DATABASE CONTEXT

Let a database $\mathcal{D}$ be associated with three kinds of context information that a Text-to-SQL system may consume:

- the **schema** $S$, comprising tables and their typed columns and key relationships;
- the **column-level semantics** $M$, comprising natural-language column descriptions, types and sample values;
- the **external knowledge** $E$, comprising data dictionaries, metric definitions and business documents.

The raw database context is denoted $D = (S, M, E)$. Given a question $q$, an end-to-end Text-to-SQL system $\Pi$ produces a SQL query

$$\hat{y} = \Pi\big(q, D\big).$$

#### B.1.2 THE COMPRESSION PROBLEM

A database compressor is a map

$$\mathcal{C} : \; D \;\longmapsto\; D'(q) = (S', M', E'(q))$$

that rewrites database context into a more compact representation $D'$, possibly conditioned on $q$ for the external-knowledge component. Let $\mathrm{Cost}(\cdot)$ denote a measure of input cost—throughout this paper we use *input-token count*, which is also the dominant per-call billing unit for commercial LLM APIs—and $\mathrm{Util}(\Pi, D, \cdot)$ denote a measure of downstream utility (e.g., schema-linking recall or execution accuracy). The compressor is required to satisfy:

For a target downstream system $\Pi$ and a workload $\{q_i\}$, the compressor seeks to minimize $\mathrm{Cost}(D')$ while preserving or improving expected downstream utility, i.e., $\mathrm{Cost}(D') \ll \mathrm{Cost}(D)$ and $\mathrm{Util}(\Pi, D', q_i) \geq \mathrm{Util}(\Pi, D, q_i)$.

Three properties of this compression problem are worth emphasizing.

**(i) Asymmetric in $q$.** The structural part $S'$ and the semantic part $M'$ are produced by query-agnostic rewrites: they depend only on $D$ and can be precomputed and cached at the database side. The external-knowledge part $E'$ is the only query-aware component and is computed online. This asymmetry is the basis for the two-phase architecture in Sec. 3.2.

**(ii) Exact preservation of structural candidates.** DBCC discards no base table or column. Structural compression uses shared components and template inheritance that can be deterministically expanded, so downstream linking retains the complete structural candidate space. Semantic componentization is

exact only relative to the extracted tag sets; the earlier conversion from free-form descriptions to tags and the online purification of documents are lossy transformations.

**(iii) Inequality in utility, not equality.** Equation equation 1 permits strict improvement because compressing redundant or noisy context often *improves* downstream accuracy by removing distractors—a phenomenon consistent with the *Lost-in-the-Middle* effect (Liu et al., 2024).

### B.2 The DBCC Framework

This section introduces a common component-accounting abstraction for the structural and normalized-semantic layers, followed by a separate query-aware treatment of external documents.

#### B.2.1 The SGCF principle

Section 3.2 now contains the formal information-layer definition, support/width statistics, gain function and constrained SGCF objective. Here we clarify its scope and implementation assumptions. The abstraction applies to query-agnostic structural signatures and normalized semantic tags; it does not treat question-dependent document relevance as an offline frequent-component problem. The unit-cost gain is used only to rank admissible candidates, whereas every reported token count is computed from the final tokenizer-specific serialization. Layer-specific constraints then determine which candidates are legal: structural components obey overlap and per-table reference limits, template edges obey relative-delta limits, and semantic components are disjoint in normalized-tag space. These constraints make the objective a practical greedy design rule rather than an optimality or approximation claim.

#### B.2.2 Architecture

DBCC realizes this design in a two-phase architecture (Fig. 2). The split has three direct consequences: (C1) *Amortization*—the dominant LLM-driven step (semantic keyword extraction) lives in Phase I, so its cost is paid once per database; (C2) *Reusability*—Phase I produces a self-contained compressed view consumable by different downstream pipelines; and (C3) *Statefulness vs. statelessness*—Phase I is a stateful database asset, whereas Phase II is stateless and query-dependent. Although the operators write to different fields of the representation, their downstream effects need not be independent because all fields influence the same linking decision. Section 4.1 therefore reports their empirical interactions without assuming orthogonality.

### B.3 Structural Compression

The structural layer addresses redundancy in the schema itself. DBCC instantiates two operators that together cover the two dominant redundancy patterns of real schemas.

#### B.3.1 Component factorization for wide tables

**Pattern** A *column group* is a set of column signatures $X \subseteq \mathcal{V}_S$ that appears in many tables. Examples include audit columns (`created_at`, `updated_at`, `updated_by`), monetary status columns (`amount`, `currency`, `status`), and standardized location attributes. Repeating $X$ in every table that contains it costs $|X| \cdot s(X)$ tokens; expressing $X$ once as a shared component and inserting a reference in every supporting table costs $|X| + s(X)$.

**Operator** For each candidate column group $X$, DBCC computes its support $s(X)$, width $w(X) = |X|$ and gain equation 2, and accepts $X$ as a component iff $s(X) \geq \tau_s$, $w(X) \geq \tau_w$ and $\text{gain}(X) \geq \tau_g$. To keep each table's rendering readable when many components apply, we additionally enforce a *per-table component quota* $K$ (the maximum number of references attached to any single table; we use $K = 8$ throughout).

**Algorithm** Naive enumeration of column subsets within a wide table is exponential. DBCC avoids this with an inverted-index trick that exploits the structure of column groups. Complete pseudocode is provided in Appendix D (Algorithm 1).

**Why the inverted index** A wide table can have hundreds of columns, so unrestricted subset enumeration is intractable. We group columns with the same exact support set $\phi(c)$ and score each equivalence class as one candidate. At a fixed support, including the whole class maximizes the proxy gain, which makes this restriction efficient. It is nevertheless a heuristic: columns with different support sets can still form a useful group over the intersection of those supports, and such nested-support groups may be missed. We choose the restricted candidate family for tractability and do not claim that it maximizes Equation equation 3. Accepted components are rewritten as references rather than removed, so the structural transformation remains reversible.

### B.3.2 Template hierarchy for homogeneous partitions

**Pattern** A *template* is a set of physical tables $\{T_1, \ldots, T_k\}$ whose column-set is identical. Time-, region- or version-sharded tables are templates by construction. A *derived template* is a template whose column-set is a superset of another template's column-set, differing only by a small delta.

**Operator** DBCC compresses a template family by recording the parent template once, plus, for each derived template, only the delta. Inheritance gives a compact tree-of-tables representation that resembles object-oriented inheritance. The complete construction appears in Appendix D (Algorithm 2).

For an inheritance link $B_i \to B_j$, we define $\mathrm{gain}(B_i \to B_j)$ as the unit-cost size of the raw signatures replaced by the link minus the unit-cost size of the parent reference and stored delta. As with Equation equation 2, this is a selection proxy; all reported token counts are measured after actual serialization.

**Design rationale and complementarity with factorization** Two thresholds control the rendering: the minimum parent size $\tau_p$ ensures that the parent carries enough structural information to be informative, and the relative-delta threshold $\tau_r$ rejects derivations whose increment is too large to read as a small extension—in pilot studies, removing $\tau_r$ produced "inheritance" relations that were technically valid but unreadable, hurting downstream linking despite reducing tokens. Factorization handles *horizontal* redundancy (columns shared across tables) while template inheritance handles *vertical* redundancy (entire tables sharing the same shape); the two operate on different carriers and select from disjoint candidate spaces, and their interaction is evaluated in Sec. 4.

## B.4 Semantic Compression

Even after structural compression, each column still carries verbose semantic context: a natural-language description, a type, and sample values. In real databases these descriptions are often highly redundant across columns—"timestamp of the last update", "timestamp of the most recent update", "last-update epoch milliseconds" are essentially the same tag. The semantic layer of DBCC compresses this redundancy in two stages.

### B.4.1 Hierarchical keyword extraction

For each column $c_i$ with description $d_i$, type $t_i$ and sample values $v_i$, DBCC produces a small set of normalized keywords

$$K_i = \mathrm{Norm}\big(K_{\text{desc}}(d_i, v_i) \cup K_{\text{type}}(t_i)\big).$$

The keyword set is intended to act as a *semantic tag* rather than a paraphrase: it should support disambiguation in schema linking, not reproduce the natural-language description.

To preserve cross-table consistency, DBCC extracts keywords *hierarchically*: at the database level when the input fits, then falling back to table level, then to chunk level, with explicit propagation of vocabulary between levels. Database-level extraction is essential: it ensures that synonymous concepts in different tables receive the same tag, which is the single largest source of subsequent component reuse. Normalized type tags such as `type:variant` and `type:struct` are exposed explicitly, so that downstream models can reason about composite columns without parsing nested type expressions.

### B.4.2 SHARED SEMANTIC-TAG COMPONENTIZATION

After hierarchical extraction, small subsets of keywords are observed to appear across many columns. The semantic-component operator applies SGCF (Eq. equation 2) to the keyword layer, with carriers $\mathcal{O}=\{c_i\}$ and information units drawn from the keyword vocabulary $\mathcal{V}_M$, so $s(X)$ here counts the number of columns whose keyword set contains $X$. DBCC accepts $X$ as a semantic component iff $s(X)\geq\tau_s$, $w(X)\geq\tau_w$ and $\mathrm{gain}(X)\geq\tau_g$. Each column $c_i$, originally tagged with $K_i$, is rewritten as a pair

$$\widehat{K}_i = (R_i,\, U_i), \qquad R_i \subseteq \mathcal{C},\;\; U_i = K_i \setminus \bigcup_{X\in R_i} X,$$

where $R_i$ is the set of references from $c_i$ to accepted components and $U_i$ is the residual keyword set carried inline by $c_i$.

**Frequency-priority component selection** Unlike structural columns, single-column keyword sets are short, so candidate components $X \subseteq K_i$ can be enumerated directly up to a width cap $w_{\max}$. DBCC then selects components by a *frequency-priority greedy* sweep: sort all admissible candidates by support $s(X)$ (with $\mathrm{gain}(X)$ and $w(X)$ as tie-breakers), and accept each candidate in turn iff it (a) still satisfies the support/width/gain thresholds against the current state and (b) is disjoint on its keyword tokens from every previously accepted component. After each acceptance the residual keyword sets are refreshed and the remaining candidates are re-evaluated. The intuition is that a high-support component produces a thicker shared backbone for the components that follow, so paying its definition cost first amortizes best across the database; the loop terminates after at most $O(|\mathcal{V}|)$ acceptances. Appendix D states the procedure formally in Algorithm 3.

**Why semantic compression cannot be replaced by structural compression** The two layers select components from disjoint vocabularies: structural components come from $\mathcal{V}_S$ (column signatures), semantic components come from $\mathcal{V}_M$ (keyword tokens). A column group such as `(created_at, updated_at, status)` is a structural component; a tag such as {`datetime, audit, mutation`} attached to the column `updated_at` is a semantic component. They co-exist in the compressed view, and the ablation in Sec. 4.1 confirms additive utility.

## B.5 EXTERNAL-KNOWLEDGE COMPRESSION

External documents—data dictionaries, metric glossaries, domain notes—contain information that is essential for resolving ambiguous columns and complex business rules, but per question only a small fraction of any given document is relevant.

### B.5.1 QUESTION-RELEVANT EVIDENCE PURIFICATION

DBCC treats the external-knowledge layer as the only *query-aware* component. Given a question $q$ and a document $d$, DBCC extracts evidence

$$E'(q) = \mathrm{Purify}(q, d),$$

designed to support schema linking and SQL generation rather than to summarize the document as a whole.

The purification procedure is a thin LLM-driven step that performs typed extraction on $(q, d)$ followed by a deterministic schema-anchoring pass; the full prompt template, the JSON schema each atom must satisfy, and the rule-based fallback that triggers when the LLM emits a malformed atom are all released with our code (see the URL in the abstract). Each piece of evidence it returns is a *typed atom* bound to a specific decision the downstream system has to make when generating SQL. We use four atom types, which together form a minimal cover of where external documents intervene in SQL generation—*which* columns/tables are used, *how* they are combined, *which* values they are filtered by, and *what* question-side vocabulary even refers to:

- **Concept-to-element mappings.** A natural-language concept in $q$ is bound to a concrete schema element, driving schema linking. E.g. "net revenue" → `REVENUE.NET_AMT`.
- **Metric definitions.** A business metric named in $q$ is expanded into the expression that computes it, driving aggregation and projection. E.g. *MoM growth* = (cur − prev)/prev.

- **Value constraints.** The encoding of a filter value is recovered so the literal in the `WHERE` clause is correct. E.g. `STATUS` is encoded as `1=created`, `2=paid`, `3=completed`, `4=refunded`, so the predicate "completed" must be written as `STATUS=3`, not `STATUS='completed'`.
- **Domain definitions.** A term in $q$ that has no direct schema counterpart is expanded into a set of schema-visible elements. E.g. "EU" $\rightarrow$ {`DE, FR, IT, ES, NL, ...`} over `COUNTRY_CD`.

**Why purification, not retrieval** A retrieval-only baseline returns text passages, which still need to be parsed in context and consume non-trivial tokens. Purification produces *typed evidence atoms* that are easier for downstream linking to consume. This is also the layer at which SGCF degenerates: the carriers (document chunks) are query-specific, so support is computed over the set of question-relevant chunks rather than all documents. Purification can be viewed as a Text-to-SQL-specific instantiation of retrieval-augmented generation, with the crucial difference that purified targets are *schema-aware*—each evidence atom is anchored to a database element, making it directly usable by schema linking. DBCC can also be combined with arbitrary retrievers (e.g., BM25 (Robertson & Zaragoza, 2009) or dense retrievers (Reimers & Gurevych, 2019)) as a pre- or post-processing step.

### B.6 Integration with Text-to-SQL Pipelines

Because DBCC rewrites only the database side, it integrates with a wide spectrum of downstream systems through a single contract: replace the system's view of $D$ with the compressed view $D'$.

#### B.6.1 Insertion points

Within a typical Text-to-SQL pipeline DBCC can be inserted at two points, mirroring the two stages of the pipeline itself—schema linking and SQL generation. The two insertion points address *different* bottlenecks of the downstream system, and DBCC contributes a different aspect of $D'$ to each.

- **Pre-linking (*shrinks the candidate space* and *sharpens token-level alignment*).** The system must decide which database elements are relevant to the question. We consider schema linking broadly, covering both granularities: token-level alignment over the full schema (Wang et al., 2020; Li et al., 2023a; Lin et al., 2020; Cao et al., 2024) and retrieval-style subsetting that narrows a far larger candidate space (Kothyari et al., 2023; Wang et al., 2025b; Li et al., 2024a; Ma et al., 2025). DBCC contributes to both. At the size axis, replacing $S$ with the structurally compressed $S'$ collapses hundreds of sharded variants into one canonical instance and exposes each repeated audit/status block once instead of dozens of times; the linker's input is smaller, and its recall ceiling rises because near-duplicate candidates that previously competed for the top-$k$ slots are now merged. At the density axis, hierarchical semantic tags in $M'$ replace verbose, inconsistently worded column descriptions, and purified evidence atoms in $E'$ provide explicit NL$\rightarrow$element anchors; the linker sees fewer, more discriminative cues per candidate.
- **Pre-generation (*reduces prompt cost and agent loops*).** If the system directly prompts an LLM (Dong et al., 2023; Pourreza & Rafiei, 2023; Zhang et al., 2023a; Gao et al., 2024; Li et al., 2024b; Pourreza et al., 2024) or runs an agentic flow (Talaei et al., 2024; Deng et al., 2025; Wang et al., 2025a; 2026), $D'$ shortens the prompt, focuses attention on discriminative information, and—for agents—cuts the number of database-exploration steps needed before the model commits to a plan.

The two insertion points compose: a system that does linking and LLM generation in series can ingest $D'(q)$ once and benefit at both stages, because $S', M', E'(q)$ are produced as a single self-contained view.

#### B.6.2 Cost model and minimal assumptions

DBCC does not modify the parser, decoder, schema-linking module, agent controller or post-processor of the downstream system: the rewrite is reversible (any system can expand $S'$ back to $S$), no additional supervision is needed (structural/semantic compression is unsupervised, evidence purification needs only the question), and no per-query online learning state is kept on the database side beyond the cached Phase I view. Let $c_{\mathrm{I}}(\mathcal{D})$ be the offline cost of Phase I, $c_{\mathrm{II}}$ the per-query cost of Phase II, $\Delta c$ the per-query token saving, and $N$ the expected number of queries against $\mathcal{D}$. DBCC

is profitable iff $N \cdot \Delta c \geq c_{\mathrm{I}}(\mathcal{D}) + N \cdot c_{\mathrm{II}}$, so the break-even number of queries is

$$N^* \;=\; \left\lceil c_{\mathrm{I}}(\mathcal{D}) \,/\, (\Delta c - c_{\mathrm{II}}) \right\rceil .$$

$N^*$ is small whenever $\Delta c \gg c_{\mathrm{II}}$, which is the regime our compressors target; we measure $N^*$ empirically in Sec. 4.

# C Experimental Setup

**Benchmarks** We use **Spider 2.0-Snow** (Lei et al., 2024) as our primary benchmark. It is a real-workflow benchmark on Snowflake with 152 enterprise databases, 547 questions and an average of 812.1 columns per database; the average gold SQL involves 161.8 tokens and 6.8 function calls. The column-count distribution is heavy-tailed: while the mean is dominated by a long tail of small databases, a substantial minority sits in the $\geq 10^4$ regime that motivates this work. Following Sec. 2.2, we partition the databases into three buckets by column count: Small ($<$1,000 columns; 76 DBs), Medium (1,000–10,000 columns; 53 DBs) and Large ($\geq$ 10,000 columns; 23 DBs). For cross-benchmark generalization we use the full **BIRD** (Li et al., 2023b) benchmark—all of its 95 databases and 1,534 development questions, with no column-count filtering.

**Models** We use **DeepSeek-V3.2** as the primary LLM for both Phase I (semantic keyword extraction, evidence purification) and the downstream task (schema linking, SQL generation). We additionally evaluate cross-LLM robustness with **GPT-4o** and **Claude-Opus-4.7**.

**Tasks and metrics** The primary task is schema linking: given a database context and a question, the LLM must output the set of schema elements (tables and columns) needed to answer the question. All metrics below follow the conventions established in recent schema-linking work—in particular RSL-SQL (Cao et al., 2024) and Apex-SQL (Cao et al., 2026)—so that our numbers are directly comparable to theirs. We report:

- **Strict Recall Rate (SRR)**: per-instance indicator of whether the prediction covers *all* gold schema elements.
- **Non-Strict Recall (NSR)** and **Non-Strict Precision (NSP)**: micro-level recall and precision over predicted vs. gold elements.
- **Token cost (Tok)**: average input tokens per request; reported in K (thousand) or M (million).
- **Execution Accuracy (EX)**: for end-to-end Text-to-SQL experiments, the fraction of generated SQL queries whose execution result matches the gold query.

**Default DBCC configuration** Structural compression uses *template hierarchy*, semantic compression uses the frequency-priority component selection of Algorithm 3 with thresholds $(\tau_s, \tau_w, \tau_g) = (2, 2, 1)$, and external-knowledge compression uses the question-driven evidence purification described in Sec. 3.5.

# D Algorithmic Details

The following algorithms provide the complete pseudocode for the three component-selection procedures summarized in Section 3.

# E Additional Experiments

## E.1 Structural Compression

We first evaluate the two structural operators in isolation. Table 7 reports schema-linking utility and end-to-end EX, against the uncompressed schema, on the three database buckets. The two informative axes are the input cost (Tok) and the resulting utility (SRR/NSR/NSP/EX); their non-monotone trade-off is examined in Sec. E.6.

The two operators capture complementary redundancy patterns and select from disjoint candidate spaces (single tables for factorization vs. table clusters for hierarchy). Component factorization is

**Algorithm 1** Column-group factorization

$\phi(c) \leftarrow \{T \in \mathcal{T} : c \in T\}$ for every column signature $c$.
$\mathcal{G} \leftarrow \{ X_S = \phi^{-1}(S) : S \subseteq \mathcal{T} \}$ (one candidate per equivalence class of $\phi$; see below).
$\mathcal{G}^\star \leftarrow \{ X \in \mathcal{G} : s(X) \geq \tau_s,\ w(X) \geq \tau_w,\ \mathrm{gain}(X) \geq \tau_g \}$.
Sort $\mathcal{G}^\star$ by $(\mathrm{gain}(X), s(X), w(X))$ in decreasing order.
$\mathcal{C} \leftarrow \emptyset$.
**for** $X \in \mathcal{G}^\star$ in sorted order **do**
    **if** $X$ disjoint from every $Y \in \mathcal{C}$ **and** per-table quota $K$ respected **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{X\}$.
    **end if**
**end for**
**Output:** $S' = \{ T \mapsto (\mathcal{R}(T),\ U(T) \setminus \bigcup_{X \in \mathcal{R}(T)} X) \}_{T \in \mathcal{T}}$, with $\mathcal{R}(T) = \{X \in \mathcal{C} : X \subseteq U(T)\}$.

**Algorithm 2** Template hierarchy construction

$\mathcal{B} \leftarrow \{ B_i \}$, where $B_i$ groups physical tables sharing the same column signature $\mathrm{Sig}(B_i)$.
Sort $\mathcal{B}$ by $|\mathrm{Sig}(\cdot)|$ in increasing order. **Process** $B_i$ **in this order**, so that any candidate parent $B_j$ with $\mathrm{Sig}(B_j) \subseteq \mathrm{Sig}(B_i)$ has already been admitted (or rejected) by the time $B_i$ is visited; when no such $B_j$ exists, $B_i$ is left as a root.
For each $B_i$, define

$$\mathrm{Par}(B_i) = \arg \max_{B_j :\, \mathrm{Sig}(B_j) \subseteq \mathrm{Sig}(B_i),\ |\mathrm{Sig}(B_j)| \geq \tau_p} \big|\mathrm{Sig}(B_j)\big|,$$

and $\Delta_i = \mathrm{Sig}(B_i) \setminus \mathrm{Sig}\big(\mathrm{Par}(B_i)\big)$.
Accept the inheritance link $B_i \to \mathrm{Par}(B_i)$ iff $\mathrm{gain}(B_i \to \mathrm{Par}(B_i)) \geq \tau_g$ and $|\Delta_i| / |\mathrm{Sig}(B_i)| \leq \tau_r$; otherwise mark $B_i$ as a root.
**Output:** forest $\mathcal{F} = \{(B_i, \mathrm{Par}(B_i), \Delta_i)\}$ of accepted links plus the roots.

stronger on SMALL and LARGE databases (where audit/status groups dominate), template hierarchy on MEDIUM (where sharded families dominate). On SMALL the gain is essentially in cost: tokens drop by $\sim 22\%$ while SRR/NSR/NSP stay within 1–3% of the baseline. On MEDIUM, template hierarchy already dominates: tokens drop by 82.0%↑ (166.0K → 29.9K) and SRR rises by 5.2%↑, NSR by 14.1%↑. On LARGE the contrast is striking: the uncompressed schema does not fit and yields SRR = 0.0%, while template hierarchy reaches 47.8% SRR with a 96.1%↑ token reduction—compression is a *prerequisite* for the model to attempt the task at all.

**Per-database-type analysis** Classifying the 152 databases by structural fingerprint into WIDE-HEAVY (max column-signature support $s_{\max} \geq 10$ and table-cluster ratio $< 0.3$; 58 DBs), PARTITION-HEAVY (table-cluster ratio $\geq 0.3$; 41 DBs) and MIXED (the remaining 53 DBs) confirms what the design predicts. Factorization wins on WIDE-HEAVY (SRR 52.1 vs. 49.0 for hierarchy; EX 15.4 vs. 13.7); hierarchy wins on PARTITION-HEAVY (SRR 53.6 vs. 46.2; EX 16.8 vs. 11.5); on MIXED neither subsumes the other and applying both is preferable (SRR 51.9, EX 15.1). Hence the two operators select components from disjoint candidate spaces and are jointly needed.

### E.2 SEMANTIC COMPRESSION

Table 8 reports schema-linking utility and end-to-end EX of semantic compression vs. the raw representation, on the same three buckets. Tokens drop by 46.5% on SMALL (9.6K → 5.1K) and by roughly 73% on both MEDIUM and LARGE (166.0K → 43.6K, 2.6M → 712.4K), confirming that semantic redundancy scales with the number of columns rather than with the size of any individual description. Utility stays close to the raw baseline on SMALL (where it is already saturated) and *rises* on the larger buckets (SRR 44.8 → 49.0 on MEDIUM; 0.0 → 8.7 on LARGE). Semantic-only compression is not sufficient to make LARGE databases tractable in isolation; the full benefit emerges only when combined with structural compression (Sec. 4.1).

**Algorithm 3** Frequency-priority semantic-component selection

**Input:** columns $\{c_i\}$ with keyword sets $\{K_i\}$; thresholds $(\tau_s, \tau_w, \tau_g)$.
Enumerate candidate components $\mathcal{X} \leftarrow \{X \subseteq K_i : |X| \leq w_{\max}\}$.
For each $X \in \mathcal{X}$, compute $s(X), w(X), \mathrm{gain}(X)$ from equation 2.
Sort $\mathcal{X}$ by $\big(s(X), \mathrm{gain}(X), w(X)\big)$ in decreasing order.
$\mathcal{C} \leftarrow \emptyset$.
**for** $X \in \mathcal{X}$ in sorted order **do**
    **if** $s(X) \geq \tau_s$ **and** $w(X) \geq \tau_w$ **and** $\mathrm{gain}(X) \geq \tau_g$ **and** $X$ disjoint from every $Y \in \mathcal{C}$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \cup \{X\}$; remove $X$ from each affected $K_i$.
        Recompute $s(\cdot), w(\cdot), \mathrm{gain}(\cdot)$ for the candidates that overlap any modified $K_i$.
    **end if**
**end for**
**Output:** component set $\mathcal{C}$; per-column residuals $U_i \leftarrow K_i \setminus \bigcup_{X \in \mathcal{C}} X$.

Table 7: Structural compression on Spider 2.0-Snow: schema-linking utility and end-to-end EX, vs. the uncompressed schema. LLM: DeepSeek-V3.2. SRR/NSR/NSP/EX in %; tokens in K or M. Best per metric per bucket in **bold**, second-best underlined.

| | SMALL | | | | | MEDIUM | | | | | LARGE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | SRR | NSR | NSP | Tok | EX | SRR | NSR | NSP | Tok | EX | SRR | NSR | NSP | Tok | EX |
| Raw schema | **61.1** | 84.3 | **78.6** | 9.6K | 28.6 | 44.8 | 67.3 | 57.1 | 166.0K | 9.4 | 0.0 | 2.9 | 50.0 | 2.6M | 0.0 |
| Comp. factor. | 58.3 | 83.0 | 78.3 | **7.4K** | 29.5 | 49.0 | 78.0 | 48.4 | 33.8K | 15.8 | 43.5 | 67.6 | 63.7 | 82.9K | 10.7 |
| Templ. hier. | 60.6 | 83.8 | 75.5 | 7.5K | **30.4** | **50.0** | **81.4** | **57.4** | **29.9K** | **17.2** | **47.8** | **69.5** | **71.6** | 100.7K | **11.6** |

### E.3 EXTERNAL-KNOWLEDGE COMPRESSION

We evaluate evidence purification on the subset of Spider 2.0-Snow questions that reference an external document. Table 9 reports the resulting schema-linking utility and end-to-end EX on the same subset.

At the *prompt* level the token savings from purification are essentially negligible—229.2K → 228.1K input tokens—because the external document is only one of several prompt components and the rest (schema, semantic descriptors, in-context examples) dominates. Despite this, replacing raw documents with purified evidence improves SRR by 2.7 percentage points, NSR by 14.7 percentage points and NSP by 7.2 percentage points on the documented subset, with a corresponding gain of 2.6 EX percentage points. Thus, this operator is better understood as an evidence-focusing transformation than as a major source of token compression: business rules, concept-to-element mappings and value constraints become explicit, schema-anchored atoms rather than being scattered across long prose.

### E.4 CROSS-LLM ROBUSTNESS

We evaluate DBCC under two additional commercial LLMs, **GPT-4o** and **Claude-Opus-4.7**, on the same Spider 2.0-Snow setting (Table 10). Two patterns emerge across both models. (i) On SMALL databases, where the uncompressed prompt already fits and is comprehensible, the relative gain shrinks because larger models already handle redundancy gracefully. (ii) On MEDIUM and especially LARGE databases, where the issue is whether the prompt fits at all, the gap widens—a 2.6M-token raw prompt remains beyond the input budget of every LLM we tested, so absolute SRR stays at 0 without compression and rises sharply once DBCC is plugged in. Claude-Opus-4.7 is the strongest of the three on raw inputs and remains the strongest under DBCC, but the relative improvement that DBCC contributes is essentially constant across LLMs, supporting our framing: DBCC addresses an input-side problem and its benefit is largely independent of downstream model capacity.

### E.5 CROSS-BENCHMARK GENERALIZATION

We evaluate DBCC on the full **BIRD** benchmark (Table 11), with no column-count filtering. We use GPT-4o as the LLM here (rather than DeepSeek-V3.2 as in the rest of the experiments) because

Table 8: Semantic compression on Spider 2.0-Snow: schema-linking utility and end-to-end EX, vs. the raw semantic representation. LLM: DeepSeek-V3.2. SRR/NSR/NSP/EX in %; tokens in K or M. Best per metric per bucket in **bold**.

| | SMALL | | | | | MEDIUM | | | | | LARGE | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Method | SRR | NSR | NSP | Tok | EX | SRR | NSR | NSP | Tok | EX | SRR | NSR | NSP | Tok | EX |
| Raw semantic | **61.1** | 84.3 | **78.6** | 9.6K | 28.6 | 44.8 | 67.3 | **57.1** | 166.0K | 9.4 | 0.0 | 2.9 | 50.0 | 2.6M | 0.0 |
| Keyword comp. | 60.6 | **85.0** | 74.8 | **5.1K** | **29.1** | **49.0** | **76.4** | 47.5 | **43.6K** | **14.2** | **8.7** | **13.3** | **60.9** | **712.4K** | **2.6** |

Table 9: External-knowledge compression on the Spider 2.0-Snow subset with external knowledge. LLM: DeepSeek-V3.2. "Tok" is the prompt-level input-token count. Best per metric in **bold**.

| Method | SRR | NSR | NSP | Tok | EX |
|---|---|---|---|---|---|
| Raw documents | 49.3 | 73.9 | 69.4 | 229.2K | 12.7 |
| Purified evidence | **52.0** | **88.6** | **76.6** | **228.1K** | **15.3** |

GPT-4o is the model on which existing BIRD numbers are most commonly reported, making the *Raw* row directly comparable to the literature; the cross-LLM evidence in Sec. E.4 confirms that the $\Delta$-pattern is essentially LLM-agnostic. Although BIRD has shorter average column counts than Spider 2.0-Snow, it features richer business semantics and external knowledge, so we expect proportionally larger gains from semantic and EK compression and slightly smaller gains from structural compression.

The qualitative pattern transfers: DBCC improves all three reported utility metrics while reducing tokens. The absolute SRR increase is smaller on BIRD (7.3 vs. 22.7 percentage points on Spider 2.0-Snow). One plausible explanation is that BIRD's generally smaller schemas leave less raw structural redundancy to exploit; however, because we do not report a per-operator BIRD ablation, we do not attribute the improvement quantitatively to any particular operator.

### E.6 COST–UTILITY PARETO FRONTIER

To map out the trade-off, we sweep the thresholds of each operator and plot *input-token cost* vs. SRR on the LARGE bucket. The sampled points (Fig. 3) reveal a non-monotone curve: shrinking the prompt initially *raises* utility, peaks at Tok $\approx$ 34.7K for full DBCC (matching the row in Table 4), and drops under extreme over-compression (Tok $\lesssim$ 10K), when the compact serialization no longer gives some discriminative columns sufficient salience. This failure is distinct from the classic Lost-in-the-Middle positional effect: here the principal risk is representation-induced information or salience loss.

### E.7 OFFLINE COST AND AMORTIZATION

We finally check whether the offline cost of Phase I is justified (Table 12). The Phase I cost is dominated by LLM-based hierarchical keyword extraction (Sec. 3.4); the structural rewrite is combinatorial and negligible, and Phase II adds a small per-question cost.

The break-even $N^*$ is small: even on SMALL databases, $\sim$4 questions already amortise the offline cost; on MEDIUM/LARGE databases $N^* = 1$, i.e. DBCC is net-positive after a single query, since $\Delta c$ scales with database size while $c_{\mathrm{II}}$ does not.

### E.8 FAILURE-MODE ANALYSIS

We finally take stock of where DBCC fails. Inspecting the schema-linking errors of the full DBCC configuration on Spider 2.0-Snow yields four recurring failure modes, whose distribution is shown in Fig. 4.

**Mode 1** arises when an audit-style column that happens to be discriminative for a given query is folded into a frequent component; the column is still recoverable by expansion, but the compressed view

Table 10: Cross-LLM robustness of DBCC on Spider 2.0-Snow. Within each LLM, best per metric per bucket in **bold** (higher is better for SRR/EX, lower for Tok).

| LLM | Method | SMALL | | | MEDIUM | | | LARGE | | |
|---|---|---|---|---|---|---|---|---|---|---|
| | | SRR | Tok | EX | SRR | Tok | EX | SRR | Tok | EX |
| DeepSeek-V3.2 | Raw | 61.1 | 9.6K | 28.6 | 44.8 | 166K | 9.4 | 0.0 | 2.6M | 0.0 |
| DeepSeek-V3.2 | + DBCC | **61.7** | **4.5K** | **31.2** | **56.3** | **14.3K** | **17.2** | **56.5** | **34.7K** | **11.6** |
| GPT-4o | Raw | 65.4 | 9.6K | 32.1 | 49.6 | 166K | 11.8 | 0.0 | 2.6M | 0.0 |
| GPT-4o | + DBCC | **65.8** | **4.5K** | **33.5** | **60.1** | **14.3K** | **19.6** | **60.7** | **34.7K** | **13.4** |
| Claude-Opus-4.7 | Raw | 67.2 | 9.6K | 33.8 | 51.4 | 166K | 12.6 | 0.0 | 2.6M | 0.0 |
| Claude-Opus-4.7 | + DBCC | **67.6** | **4.5K** | **35.1** | **62.3** | **14.3K** | **20.7** | **63.1** | **34.7K** | **14.5** |

Table 11: Cross-benchmark generalization of DBCC on Spider 2.0-Snow and the full BIRD development set. LLM: GPT-4o. Within each dataset, best per metric in **bold** (higher is better for SRR/NSR/NSP/EX, lower for Tok).

| Dataset | Method | SRR | NSR | NSP | Tok | EX |
|---|---|---|---|---|---|---|
| Spider 2.0-Snow | Raw | 38.2 | 56.6 | 55.7 | 925K | 14.6 |
| Spider 2.0-Snow | + DBCC | **60.9** | **81.0** | **67.4** | **18.0K** | **22.3** |
| BIRD | Raw | 53.2 | 70.4 | 60.8 | 38.6K | 53.6 |
| BIRD | + DBCC | **60.5** | **77.1** | **63.9** | **12.4K** | **58.2** |

does not surface it as salient. **Mode 2** reflects the limits of LLM-driven keyword extraction: synonyms can survive normalization at the table level but disagree across tables, hurting cross-table linking. **Mode 3** is upstream of DBCC and is shared by any external-knowledge-aware system. **Mode 4** is specific to the template hierarchy operator: when the parent template is large, the per-derived $\Delta$ may include composite columns that are easy to overlook.

### E.9 SUMMARY, LIMITATIONS AND FUTURE WORK

We summarise the answers to the five questions of Sec. 4. **(Q1)** On LARGE databases DBCC cuts linker input by $98\%$ and makes an otherwise over-budget condition executable, reaching $56.5\%$ SRR with DeepSeek-V3.2 and $63.1\%$ with Claude-Opus-4.7; under GPT-4o, end-to-end EX on the full Spider 2.0-Snow evaluation rises from $14.6\%$ to $22.3\%$. **(Q2)** The operators exhibit both complementarity and antagonism; structural compression is the main enabler at large scale, while the joint configuration performs best overall. **(Q3)** The observed pattern transfers across the evaluated LLMs, benchmarks and three downstream systems; stacking experiments with Crush4SQL and LinkAlign yield further gains, and the token-based Phase I cost amortises after one query on medium/large databases. **(Q4)** The sampled cost–utility curve is non-monotone with its best observed point at Tok $\approx$ 34.7K. **(Q5)** Discriminative-column absorption and cross-table keyword inconsistency account for $62.7\%$ of categorized residual errors, motivating confidence-aware recovery and improved normalization.

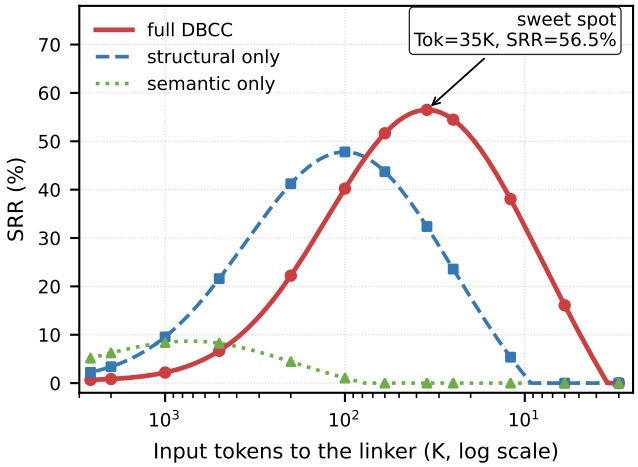



Figure 3: Cost–utility Pareto frontier on LARGE databases of Spider 2.0-Snow. LLM: DeepSeek-V3.2. Three threshold sweeps are shown: structural-only, semantic-only, and full DBCC. The horizontal axis is the input-token cost (log scale, smaller-is-better, hence inverted); the vertical axis is SRR. SRR rises non-monotonically as the prompt shrinks, peaks at Tok $\approx$ 34.7K for full DBCC, and drops at extreme over-compression.

Table 12: Offline cost and amortization of DBCC on Spider 2.0-Snow. LLM: DeepSeek-V3.2. $N^*$ = break-even number of queries per database. "P-I Tok" and "P-I s" are Phase I token cost and wall-clock time per database; "P-II Tok" is Phase II cost per question; "$\Delta$/q" is per-query saving on the downstream pipeline.

| Scale | P-I Tok | P-I s | P-II Tok | $\Delta$/q | $N^*$ |
|---|---|---|---|---|---|
| SMALL | 14K | 26 | 0.7K | 4.6K | 4 |
| MEDIUM | 138K | 240 | 1.3K | 152K | 1 |
| LARGE | 2.0M | 3,500 | 1.6K | 2.57M | 1 |

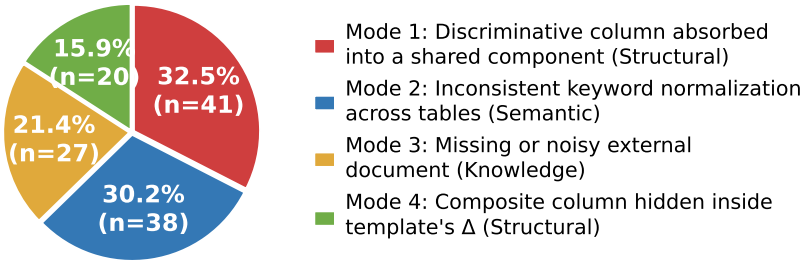



Figure 4: Distribution of failure modes of full DBCC on Spider 2.0-Snow. LLM: DeepSeek-V3.2. Slices show the share of each failure mode within the schema-linking error subset; absolute case counts in parentheses.