Comparison and Analysis of Value Linking in Text-to-SQL Systems [Experiments & Analysis]

APOSTOLOS KOUKOUVINIS*, Athena Research Center, Greece
GEORGE KATSOGIANNIS-MEIMARAKIS, Athena Research Center, Greece and Université Grenoble Alpes, France
GEORGIA KOUTRIKA, Athena Research Center, Greece

Value linking in text-to-SQL systems is the problem of associating values mentioned or implied within a NL query with their database entries. Despite existing efforts, their performance and comparative effectiveness remain unexplored. This work provides the first systematic study focused exclusively on value linking. First, we present a taxonomy that encapsulates the key steps and design choices for value linking. To enable a thorough analysis, we introduce **VLD-Bench**, a new, challenging benchmark that incorporates 18 categories of lexical and semantic discrepancies between values referenced in the queries and database entries. Our evaluation shows that state-of-the-art text-to-SQL systems suffer up to a 64% drop in execution accuracy due to value linking errors. A fine-grained analysis reveals how design choices impact robustness to linguistic variation and system latency, offering key insights and directions for future research.

CCS Concepts: • **Human-centered computing → Natural language interfaces**; • **Information systems → Structured Query Language**; **Information extraction**; **Search engine indexing**; • **General and reference → Evaluation**.

Additional Key Words and Phrases: Text-to-SQL; Value Linking

ACM Reference Format:
Apostolos Koukouvinis, George Katsogiannis-Meimarakis, and Georgia Koutrika. 2026. Comparison and Analysis of Value Linking in Text-to-SQL Systems [Experiments & Analysis]. *Proc. ACM Manag. Data* 4, 3 (SIGMOD), Article 152 (June 2026), 26 pages. https://doi.org/10.1145/3802029

1 Introduction

Text-to-SQL systems facilitate intuitive access to structured data by allowing users to express queries in natural language [20]. Recent advancements in large language models [2, 14] have greatly improved the ability of these systems to interpret user intent and generate syntactically and semantically correct SQL queries [18, 29]. A critical subtask in this process is *value linking*, which refers to the problem of associating values mentioned in a natural language query (NLQ) with their respective database entries. Accurate value linking is essential for generating executable SQL queries that faithfully reflect the user intent, particularly in the presence of lexical variation, ambiguity, or implicit references of database entries.

Despite recent progress, value linking in text-to-SQL remains a challenging task. First, the system must identify which spans in the NLQ refer to database values. For example, in the NLQ "Show me Italian restaurants in New York City", "New York City" must be recognized as a value reference.

*Corresponding author.

Authors' Contact Information: Apostolos Koukouvinis, Athena Research Center, Athens, Greece, a.koukouvinis@athenarc.gr; George Katsogiannis-Meimarakis, Athena Research Center, Athens, Greece and Université Grenoble Alpes, Grenoble, France, katso@athenarc.gr; Georgia Koutrika, Athena Research Center, Athens, Greece, georgia@athenarc.gr.

This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 2836-6573/2026/6-ART152
https://doi.org/10.1145/3802029

Second, the system must resolve ambiguity by mapping the reference to the correct column and value in the data. "New York City" should link to an entry in a `city` column, not to "New York" in a `state` column. Third, the system should handle variations in how users express the same value compared to the value stored in the data. An NLQ may use abbreviations like "NYC", synonyms like "Big Apple", or paraphrases like "The City that never Sleeps" instead of "New York City". Even minor spelling or formatting errors can cause the SQL query to fail. A robust value linking mechanism must address these issues without requiring users to know the database contents. Otherwise, the system may generate SQL statements whose `WHERE` or `HAVING` clauses reference incorrect or non-existent values, leading to erroneous results. Such mistakes undermine the system's ability to retrieve relevant information and its overall reliability, regardless of how syntactically correct the rest of the query is.

Value linking can be broadly divided into two key steps: *value reference detection*, which identifies value references (i.e., spans in the NLQ that potentially correspond to database values) and *value mapping*, which links these references to the actual database values. Text-to-SQL systems handle value linking in diverse ways. Some systems employ sophisticated techniques, including detecting value references in the NLQ using LLMs or NER [3, 36, 39] and building indexes over database values, making value mapping more efficient [17, 36]. Despite its critical role, there is no systematic study comparing these diverse approaches. Existing evaluations focus primarily on end-to-end text-to-SQL accuracy, leaving the performance and comparative effectiveness of different value linking methods largely unexplored. Moreover, the widely-used Spider [40] and BIRD [23] benchmarks mostly contain value references that are exact matches to database entries [28], limiting their ability to assess value linking performance under realistic conditions.

In this work, we conduct the first systematic study on value linking. We introduce a **Unified Taxonomy** that organizes the value linking landscape by capturing its core design patterns and implementation choices, offering a structured framework for analyzing existing methods, comparing design differences, and identifying gaps. We also conduct the first experimental evaluation of value linking mechanisms within text-to-SQL systems.

To evaluate value linking under realistic conditions with value discrepancies, we introduce **VLD-Bench** (Value Linking Discrepancy Benchmark), which systematically applies 18 categories of lexical and semantic mismatches between NLQ value mentions and their corresponding database entries. We develop a rigorous **generation methodology** combining LLM-based perturbation with manual curation to ensure quality and realism. This benchmark fills a critical evaluation gap by simulating real-world challenges such as paraphrasing, abbreviation, synonymy, and ambiguity. To ensure our findings generalize beyond established benchmarks and are not artifacts of Spider and BIRD, we apply the same generation process to an in-house text-to-SQL dataset built over the OpenAIRE Research Graph,[1] creating **an independent hold-out evaluation set**. This validation confirms that the observed fragility persists across completely unseen schemas and real-world data.

Our evaluation demonstrates that current value linking approaches face fundamental limitations when confronted with realistic linguistic variations. While systems perform well on benchmarks with exact-match values, their accuracy degrades severely on VLD-Bench – up to 64% in end-to-end execution accuracy. We further demonstrate that value linking directly affects text-to-SQL reliability, with value linking failures propagating systematically to SQL generation errors. Through systematic evaluation of common value linking designs across all 18 discrepancy categories, we identify precision-recall-latency trade-offs of different design choices. We complement this with a scalability analysis, measuring indexing time, storage overhead, and query latency as database size increases, revealing critical operational constraints for real-world deployment. From these findings,

[1]https://www.openaire.eu/

we distill concrete lessons learned and prescriptive guidelines to inform future research and system design.

In a nutshell, our contributions are the following:

- We develop a comprehensive taxonomy that breaks value linking into distinct stages, enabling systematic analysis and comparison of existing approaches.
- We provide a systematic overview and comparative analysis of state-of-the-art systems, including frontier proprietary LLMs, mapping their implementation choices to our taxonomy.
- We present VLD-Bench, a challenging value linking benchmark featuring 18 categories of lexical and semantic discrepancies.
- We validate our findings on an independent hold-out dataset by applying our generation process to an in-house text-to-SQL corpus built over the OpenAIRE Research Graph, showing that value linking fragility generalizes to entirely unseen schemas.
- Our extensive experimental study evaluates systems across all discrepancy categories, revealing how architectural choices relate to performance on specific linguistic variations, quantifying accuracy–latency trade-offs, and demonstrating the critical role of value linking on end-to-end text-to-SQL accuracy.
- We perform a scalability analysis of value linking across databases of varying sizes, measuring indexing time, storage requirements, and query latency to expose operational constraints.
- We distill concrete lessons learned and prescriptive guidelines for practitioners and researchers, highlighting the necessity of dedicated value linking mechanisms and identifying key directions for future work.[2]

2 Related Work

Early approaches to value linking relied on exhaustive scans of the database content to establish mappings between value references in the NLQ and database values [3, 24, 37]. As databases scale, however, exhaustive search quickly becomes impractical. To address this, several systems [12, 17, 22, 31, 33, 36] introduced specialized indexing mechanisms for faster value retrieval. Approaches also differ in how they identify value references in the NLQ: some map the entire NLQ directly to database values [24, 26], while others use LLMs for detecting value references [12, 31, 36, 38].

Despite advances in value linking and its growing adoption in text-to-SQL systems, a systematic study has been notably absent. Prior surveys have covered text-to-SQL broadly [18, 20, 34] and even schema linking specifically [11, 13, 21, 25], but none have focused on value linking. Unlike these prior surveys, our work provides the first dedicated systematic study, taxonomy, and in-depth evaluation exclusively on value linking, deconstructing its key steps, design choices, and performance implications.

Prior work highlighted the high lexical similarity between NLQ references and database content. To address this, recent benchmarks introduce perturbations to either the NLQs or the database. Spider-Syn [9] is a human-curated dataset derived from Spider, where table names, column names, and values in NLQs are replaced with synonyms or paraphrases, while changing the SQL. Spider-DK [10] applies domain-knowledge-aware perturbations to Spider, including five NLQ modification categories. For value linking, the most relevant is “Synonym Substitution”, where value references are replaced with synonyms. Spider-Realistic [6] removes explicit column-name mentions in a subset of Spider, focusing on WHERE-clause questions and manually replacing column names with

[2]Code available at: https://github.com/athenarc/experimental-analysis-of-value-linking

paraphrases. Dr. Spider [4] combines crowd-sourcing and model-assisted generation (human-in-the-loop) to produce a diverse set of 17 perturbation categories across DB, NLQ, and SQL spaces; its “Value-synonym” category attempts to capture value-level paraphrases.

While these works uncover important limitations in value linking, they provide only a partial treatment, predominantly emphasizing basic synonym or paraphrase substitutions without a systematic categorization of the diverse discrepancy categories that arise in real-world scenarios. Additionally, true value-discrepancy cases remain scarce in these datasets. For example, the authors of Spider-Syn [9] report only 27 examples with altered values. Our manual examination of Spider-DK [10] reveals 71 modified cases, many of which are near-duplicates; a recurring instance is replacing ‘USA’ with ‘America,’ repeated across multiple questions. Spider-Realistic [6] focuses on WHERE clauses, but its changes appear restricted to column names rather than values. Finally, although the value-synonym category in Dr. Spider [4] contains 506 pairs, our analysis shows that only 87 involve genuine synonym or paraphrase changes for string values. The remainder are numerical variations (e.g., ‘10k’ vs. ‘10,000’), which modern LLMs can typically resolve when example values are available.

In contrast, we focus exclusively on value discrepancies between NLQs and database values, introducing the first explicit and comprehensive catalog of 18 lexical and semantic discrepancy categories—including abbreviations, synonyms, paraphrases, ambiguities, and formatting variations. Unlike prior work, we design a specialized value-centric generation pipeline that produces a substantially larger and more diverse set of examples, enabling fine-grained evaluation of value linking’s robustness, efficiency, and impact on end-to-end text-to-SQL performance.

3 The Value Linking Problem

Given a natural language question (NLQ), value linking is the task of identifying and associating specific database values mentioned or implied within the query. This involves mapping the NLQ or specific phrases within it, termed *value references*, to their corresponding database entries. The output of this process is a set of value links. Each value link identifies a specific database value, along with its location (i.e., the table and column).

The essence of value linking is to inform the text-to-SQL system about the DB values that are necessary to create the requested SQL. The generated SQL will not return correct results if values are not expressed as stored (e.g., "NYC" instead of "NY"), regardless of syntactic correctness.

Effective value linking presents two primary challenges: *robustness* and *efficiency*. First, the process must be robust to lexical and semantic discrepancies, including typographical errors, formatting variations, and semantic transformations. An ideal system must accurately resolve these variations to ensure correct values in the final SQL query. Second, the process must be efficient and scalable, as it is part of an interactive pipeline.

It should also be noted that handling values in the context of text-to-SQL can involve additional tasks that are similar to value linking but have fundamental differences. These tasks present challenges such as *interpreting logical conditions* (inferring operators, thresholds, or date logic) and *resolving format mismatches*. For instance, NLQs like “sales greater than 123” require the system to identify the relevant DB column (*schema linking*) and apply the correct operator, preserving ‘123’ as a *literal threshold*. Similarly, date values often require *date-based reasoning* or *format transformation* (e.g., “January 15” might need to be transformed to “15/01”). Furthermore, a reference like “last quarter” necessitates calculating a date range based on the current time or using pre-defined *business logic* to resolve ambiguity.

These tasks and their required approaches are distinct from the task of *retrieving an existing value* from the database (i.e., value linking). In the first example, there might be no product with ‘123’ sales, and even if a product with ‘122’ sales exists, replacing it with the existing ‘122’ via a

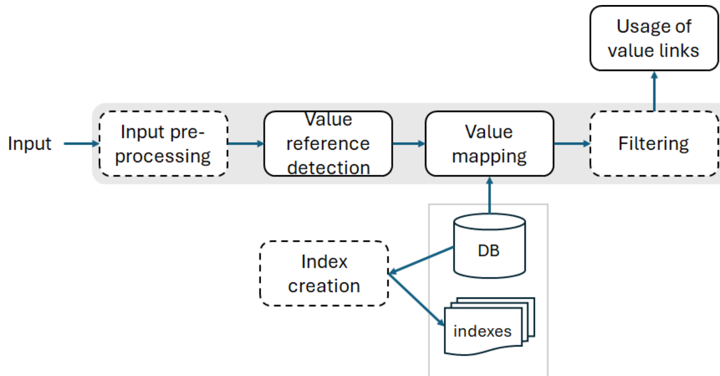

Fig. 1. An overview of the value linking process. Dashed outlines represent optional steps or inputs.

linking mechanism would yield an incorrect query. Instead, it is necessary to keep the mentioned value as is and no retrieval is required. For dates, searching database contents is similarly unhelpful, as the exact boundary dates referenced in the NLQ may not appear in the data. Although date normalization may be required to match the database format, this can be handled by providing a few random samples from the relevant column, without any retrieval step. Thus, because these numerical and date-related challenges center on logical interpretation and literal preservation rather than linking values to database entries, they fall outside the scope of *value linking*. In this work, we therefore focus on value linking for string-based data.

4 Value Linking Taxonomy

We break down value linking in text-to-SQL into key steps and analyze how different systems implement them. Figure 1 illustrates the general value linking workflow. Given the input NLQ, an optional *pre-processing* step is to simplify and clean this query. Next, the system performs *value reference detection* to identify phrases in the input that may refer to database values; in some approaches, this explicit detection step is bypassed by treating the entire NLQ as a single value reference. Then, *value mapping* searches for possible value links, either directly in the database or using indexes. Finally, an optional *filtering* step removes false value links. Next, we detail each step, explain implementation choices and present key text-to-SQL systems that use them.

4.1 Input

Value linking requires the **NLQ** and access to the **database contents** as a minimum input for searching relevant values. Some systems also use a **partial SQL sketch** (ZeroNL2SQL [15], CatSQL [8]) as an additional input—an incomplete SQL template generated by a preliminary schema-linking model. It contains the correct SQL structure, including the specific tables and columns, but uses placeholder slots for the literal values (e.g., `WHERE city = [SLOT]`). In this case, the task of value linking process is to find the correct database value to fill these slots.

4.2 Input Pre-Processing

Input pre-processing is an optional step that can enhance value linking by standardizing text and reducing noise. Some systems perform **lemmatization** (IRNet [16]), which is a technique that unifies different variations of a word by converting them to their base form, thereby improving consistency in matching. Another frequent practice is **stopword removal** (BRIDGE [24]), which

eliminates frequently occurring words (e.g., 'the', 'and', etc.) that may not contribute to identifying relevant values.

While these steps often produce cleaner and more uniform input, they can have drawbacks. Lemmatization might change words in ways that affect their intended meaning, especially in domain-specific contexts. Similarly, removing stopwords can discard value references, as they might be actual database values or part of database values. Consequently, some systems skip this step entirely, balancing the benefits of pre-processing against the risk of losing important semantic information (e.g., CodeS [17], CHESS [36]).

4.3 Value Reference Detection

This step identifies which NLQ parts may refer to database values, simplifying and speeding up value link discovery. The input is the NLQ, potentially pre-processed, and the output is a list of candidate value references. The precision–recall trade-off is crucial: overestimating value references to maximize recall can introduce false positives and increase database searches during value mapping, while underestimating them to maximize precision risks missing actual value references and impairing value linking performance.

A straightforward approach is to bypass explicit value reference detection and treat the entire NLQ as a single candidate (DART [26], BRIDGE [24]). This approach demands a sophisticated value mapping mechanism to compensate for the lack of this step. Additionally, as the NLQ lengthens, systems must manage more complex similarity assessments between full sentences and database entries.

An alternative approach that results in more comprehensive searches involves splitting the input into **n-grams** and treating each n-gram as an individual candidate (CodeS [17], IRNet [16], RAT [37]). This method ensures that no potential value reference is missed, thereby maximizing recall. However, it can introduce significant overhead in the value mapping process, as each n-gram needs to be checked against the database. This increases latency and may retrieve numerous irrelevant matches.

Named Entity Recognition (NER) (ValueNet [3]) can effectively detect value references. For instance, in the NLQ "Return hotels in New York," a NER model would identify "New York" as an entity and treat it as a value reference. This method is well-suited for databases containing named entities like people and locations. However, it does not perform as well in specialized domains.

Recent advancements in LLMs have led to systems using **LLM prompts** (CHESS [36], CHASE [31], XiYan [12], TCSR [39]) to extract value references. An LLM is given the NLQ to identify parts referring to database values. This method can generalize across domains but introduces latency, as each query requires an LLM call.

Finally, some systems enhance value reference detection by adding synonyms or alternate expressions for identified references, leveraging **external knowledge** sources, like **knowledge graphs or LLMs** (TCSR [39], IRNet [16]). While this strategy can reveal how a value is phrased in the database, it increases latency by expanding the set of candidate references for detection and mapping.

Ultimately, the choice of method depends on the size of the database and the performance requirements of the retrieval mechanism. Splitting into n-grams ensures high recall for small databases where retrieval is fast. Systems handling large databases may opt for NER or LLM-based methods, trying to increase precision for the benefit of value mapping. Treating the entire input as one reference demands more sophisticated similarity matching during retrieval, potentially making it a really slow process.

4.4 Value Mapping

During this step, the system looks up identified value references to find matching database values. This involves two components: (*a*) the *search strategy* (how to retrieve candidates), and (*b*) the *comparison metric* (how to assess similarity).

The simplest search strategy is an **exhaustive search** (e.g., BRIDGE [24], ValueNet [3]), where the entire database is scanned, ensuring complete coverage but becoming impractical as database size grows. Since value linking occurs while users await results, excessive latency severely impacts user experience.

To address this, a **hierarchical search** strategy (ZeroNL2SQL [15]) expands the search scope progressively, starting from a target column that is predicted beforehand (e.g., by a model generating a partial SQL sketch). The search begins within this predicted column; if no match is found, the scope widens to include all columns in the same table, and finally, the entire database. The process stops once a match is found, reducing computations. However, this early pruning may miss a more relevant match located elsewhere. In the worst-case scenario, the approach degrades into an exhaustive search of the entire database.

Index-based search can achieve much better performance by leveraging pre-computed **indexes** (Section 4.5), to speed up value mapping (CHESS [36], CHASE [31], XiYan [12], CodeS [17]).

Both exhaustive and hierarchical search require the system to evaluate similarity between value references and database values. The simplest method is **exact or substring matching** (TCSR [39], IRNet [16], RAT [37]). However, this approach may exclude relevant values because it treats even minor lexical variations as entirely different. **Edit distance** methods (BRIDGE [24], ValueNet [3], ZeroNL2SQL [15]) address minor discrepancies. However, beyond lexical similarity, semantic relationships between value references and database values can be captured by **embedding-based similarity** using inner product or cosine similarity (CatSQL [8], DART [26]).

Selecting a value mapping strategy requires balancing the choice of search method with the type of comparison metric used, as these decisions jointly affect efficiency and matching accuracy. Exhaustive search is practical for small databases and supports any comparison metric, but becomes prohibitive at scale. Index-based search is essential for larger systems, where the index type constrains the comparison metric: sparse indexes enable efficient lexical comparisons, while dense indexes support semantic matching. The hierarchical strategy attempts to limit search scope but risks missing relevant matches or defaulting to exhaustive search in the worst case. The final decision should account for indexing overhead, runtime latency, and the application's need for lexical versus semantic matching accuracy.

4.5 Index Creation

To mitigate runtime cost, many systems build **indexes** offline to accelerate lookup operations, dramatically improving inference efficiency. In contrast, systems that omit indexing rely on exhaustive search, which is feasible only for small datasets like Spider [40] but becomes impractical for large-scale databases.

Indexes used in value linking are categorized by their encoding method: **sparse vectors**, which encode lexical features such as token frequency or exact string matches, and **dense vectors**, which capture semantic relationships using continuous representations derived from neural models. Indexes using sparse vectors are usually faster and ensure that if a database value is lexically similar or uses similar terms with a value reference, it will be successfully located. A common choice is the BM25 index (CodeS [17], OmniSQL [22], CSC-SQL [33]), known for its speed and effectiveness in keyword searches. MinHashLSH indexing (CHESS [36], CHASE [31], XiYan [12]) approximates Jaccard similarity between two texts using MinHash signatures and Locality Sensitive Hashing.

These approaches struggle to generalize to semantic similarity between synonyms and paraphrases, and may miss some value links.

Some systems build indexes on dense vector representations generated from LLMs, leveraging their power to capture semantic similarity between value references and stored values. This approach captures paraphrases and variations well, but may miss value links due to context and domain dependency. A key design decision in dense representation is the embeddings' granularity. One approach computes *value-level embeddings* (CatSQL [8]), while *row-level embeddings* (DART [26]) generate a single vector per record, reducing stored embeddings but possibly generating more false positives. Using stored embeddings during value mapping can be accelerated with vector databases like FAISS [7, 19].

4.6 Filtering

To refine value links, some systems incorporate a filtering step to eliminate false positives from the initial search. This involves a comparison between value references from the NLQ and the value links from the value mapping step. Although this comparison can be computationally intensive, it remains tractable due to the limited candidate set. The result is a finalized list of value links.

Based on the type of metric used for filtering, the two main filtering approaches are: **lexical** (CodeS [17], OmniSQL [22]) and **semantic** (TCSR [39], OpenSearch-SQL [38]). Lexical filtering uses algorithms like longest common substring and edit distance to measure string similarity. Semantic filtering employs embedding-based similarity to capture semantic variations. Some systems use a **hybrid** approach (CHESS [36]), requiring matches to be similar both lexically and semantically.

The decision to apply a filter depends on the number of value links produced during mapping. When many links are returned, filtering helps reduce false positives and preserve precision. The choice between lexical and semantic filters depends on the mapping output. Semantic filters are better suited when mappings include semantically similar candidates, as lexical filters may wrongly discard valid matches. Conversely, lexical filters suffice when semantic variation is minimal. Filtering is especially common in systems that rely on indexes, where an additional refinement step is needed to discard false positive value links.

4.7 Usage of Value Links

Value links are integrated into the text-to-SQL pipeline. In systems using encoder-decoder models, value links enrich the input representations for the neural network. For instance, RAT [37] incorporates the links as a bias within its self-attention mechanism, explicitly signaling the connection between a question token and the matched column. Systems like ValueNet [3] and BRIDGE [24] directly augment the serialized input sequence by concatenating the identified database values alongside their corresponding columns. In contrast, systems based on LLMs typically utilize value links to augment the prompt (CodeS [17], CHESS [36], OpenSearch-SQL [38]). Finally, systems that use a partial SQL sketch fill the value slots in the sketch with the retrieved values (CatSQL [8], ZeroNL2SQL [15]).

4.8 Random Value Sampling

A distinct approach bypasses the steps above entirely. Instead of retrieving specific values based on the NLQ, this method augments the database schema provided to the LLM with a set of randomly sampled values for each column. While this is not value linking in the traditional sense, it provides the model with context regarding data formats. This directly aids in resolving format mismatches for numerical and date values (Section 3), enabling the model to infer correct data patterns implicitly.

System	Pre-Processing	Value Reference Detection	Value Mapping		Filtering
			Search strategy	Comparison metric	
IRNet [16]	Lemma-tization	N-grams augmented by a knowledge graph	Exhaustive search of column and table names	Exact and partial match	—
RAT-SQL [37]	Stopword removal	N-grams	Exhaustive search of database values	Exact and partial match	—
ValueNet [3]	—	NER and Heuristics	Exhaustive search of database values	Edit distance and heuristics	—
BRIDGE [24]	Stopword removal	All tokens as one sequence	Exhaustive search of database values	LCS and edit distance	—
CodeS [17] OmniSQL [22] CSC-SQL [33]	—	N-grams	BM25-indexed search	BM25	Edit distance
CHESS [36] CHASE-SQL [31] XIYAN-SQL [12]	—	LLM detection	MinHashLSH-indexed search	Jaccard similarity estimated via MinHash	Edit distance and Embedding cosine similarity
TCSR-SQL [39]	—	LLM detection and LLM augmentation with synonyms	Exhaustive search of database values	Exact and partial match	Embedding cosine similarity
ZeroNL2SQL [15]	—	LLM detection	Hierarchical exhaustive search of database values	Edit distance and Embedding inner product similarity	—
CatSQL [8]	—	All tokens as one sequence	Dense vector indexing within one column	Embedding cosine similarity	—
DART-SQL [26]	—	All tokens as one sequence	Dense vector indexing of database rows	Embedding cosine similarity	—
OpenSearch-SQL [38]	—	LLM detection	Dense vector indexing of database values	Embedding cosine similarity	Embedding cosine similarity

Table 1. Taxonomy of value linking in text-to-SQL systems

Contextual-SQL [1] exemplifies this strategy. Rather than employing a dedicated retrieval module to link specific spans in the NLQ to database entries, it constructs a prompt that includes foreign key relationships and a small set of distinct, randomly sampled values for every column.

5 Overview of Systems

In this section, we examine how different systems implement value linking within their text-to-SQL pipelines and relate their design choices to our taxonomy. Table 1 presents a mapping between the taxonomy dimensions and the systems that perform value linking.

IRNet [16] treats value linking as a sub-component of its schema linking process. It lemmatizes NLQ tokens during preprocessing. For value reference detection, it first generates n-grams (1-6 tokens) and attempts to match them against table and column names. Any unmatched n-grams are considered initial value references, which are then enhanced with semantically related terms from ConceptNet [35] to form an augmented set of references. For the value mapping stage, it employs an exhaustive search strategy, comparing these augmented references against table and column names using exact and partial matches as the comparison metric. The process does not include a filtering step. For the usage of value links, the resulting schema links are appended to the encoder input. This approach acts as a precursor to modern value linking, as it connects references to schema names but not to actual database values.

RAT-SQL [37] pre-processes the NLQ to remove all stopwords. For value reference detection, it considers all unigrams of the processed NLQ as candidate value references, which are then matched against the database content using exhaustive search with exact and partial matching. There is no filtering step. For the usage of value links, a successful match establishes a schema link between the reference and the column, which is then used by the encoder-decoder network for SQL generation.

ValueNet [3] starts its value linking pipeline with value reference detection. It employs a combination of methods: heuristics, such as extracting quoted values, dates, and capitalized words, and

Named Entity Recognition (NER) using both a custom transformer-based model and a commercial API. For the value mapping stage, it uses two strategies. For value references identified as common database patterns via heuristics, such as gender abbreviations ('M') or location codes ('JFK'), it maps them directly to their predefined database entries (e.g., 'M' to 'male'), avoiding a database search. For the rest of the value references, it performs an exhaustive database search, using the Damerau-Levenshtein distance as the comparison metric and keeping the top-N results that exceed a specific threshold. This threshold is adjusted based on the type of value reference, with stricter limits for quoted text and exact match requirements for dates and numbers. Finally, each value link is appended to the neural encoder's input by concatenating the identified database values alongside their corresponding columns.

BRIDGE [24] removes stopwords from the NLQ as an input pre-processing. For value reference detection, it treats the entire NLQ as a single value reference. This value reference is matched against the database content using a fuzzy matching algorithm that combines the Longest Common Subsequence (LCS) and edit distance. The process does not include a filtering step. Finally, each value link is appended to the neural encoder's input by concatenating the identified database values alongside their corresponding columns.

CodeS [17] builds a sparse BM25 index [32] over the entire database content during the index creation step. During the online input pre-processing step, it processes the NLQ by removing stopwords. For value reference detection, it extracts n-grams (1-4 tokens) from the NLQ, treating each n-gram as a value reference. For the value mapping stage, it queries the BM25 index with each value reference to retrieve an initial set of candidate value links. The comparison metric for this initial retrieval is the BM25 score. The filtering stage then refines these value links using a lexical filter combining LCS and edit distance, and only matches exceeding a similarity threshold are retained. The final set of retained value links is appended to the prompt of the LLM responsible for generating the SQL.

Notably, the recently proposed **OmniSQL** [22] and **CSC-SQL** [33] adopt the same value linking mechanism directly from CodeS. Their primary differences lie in using more powerful base LLMs and different prompting techniques, which do not affect value linking.

CHESS [36] builds a MinHashLSH [5] index over the database content during the index creation step. It takes the NLQ as input without applying any pre-processing. For value reference detection, it prompts an LLM with few-shot examples to extract keywords, keyphrases, and named entities from the NLQ, all of which are treated as value references. For the value mapping stage, it queries the MinHashLSH index with the detected references; the comparison metric is the Jaccard similarity approximated by the index. The system then applies a multi-step hybrid filtering step. First, it filters the returned value links using both the edit distance (lexical) and the cosine similarity of their embeddings (semantic) using fixed thresholds. Second, it further refines the results by grouping the remaining value links by column for each value reference. Within each group, it identifies the highest similarity scores and retains only matches that reach at least 90% of these maxima. The final set of retained value links is appended to the prompt of the LLM that is responsible for generating the SQL.

A similar value linking strategy is also employed in **CHASE-SQL** [31] and **XIYAN-SQL** [12], though their implementations are not publicly available for a direct comparison.

TCSR-SQL [39] does not create an index and does not apply any pre-processing step. For value reference detection, it uses an LLM to identify references within the NLQ and predict the specific tables and columns where they are likely to be found. The LLM also augments these original references by generating up to five synonyms or variations for each one. For the value mapping stage, it employs a targeted exhaustive search. For each value reference and its LLM-generated

variations, it performs exhaustive exact and partial matching against predicted columns, then groups all resulting value links under the original reference. For filtering, it computes embeddings for the original reference and each potential value link, retaining only the one with the highest cosine similarity. The final set of retained value links is appended to the prompt of the LLM that is responsible for generating the SQL.

ZeroNL2SQL [15] employs a two-stage process for text-to-SQL generation, where small language models first generate multiple partial SQL sketches - SQL query templates that define the correct structure (tables and columns) but leave empty slots for the literal values. An LLM then iteratively attempts to complete each sketch by filling in the missing values. The process starts with the LLM filling a sketch's value slots directly from the NLQ, without explicit linking. If execution fails or returns no results, a refinement loop begins: the offending literal becomes the value reference, and a hierarchical search maps it to database values-first within the sketch's columns, then the table, then the entire database. Similarity is configurable, either lexical (edit distance) or semantic (word2vec, BERT) similarity. At each level, if a match exceeds a threshold, the LLM regenerates and re-executes the SQL. The loop continues until a query succeeds or the hierarchy is exhausted, after which the system moves to the next candidate sketch.

CatSQL [8] uses a sketch-based text-to-SQL pipeline where predefined templates are filled by a neural model. For value indexing, it precomputes column-wise dense embeddings of all distinct database values using word2vec [27]. The value linking stage takes the NLQ and a partial SQL sketch that specifies tables and columns but leaves value slots blank. No pre-processing is applied. For value-reference detection, CatSQL treats the entire NLQ as a single reference. Value mapping performs an index-based search over embeddings of the target column specified by the sketch, using cosine similarity. The system selects the single best match only if its score exceeds a predefined threshold; otherwise, it falls back to the literal from the NLQ to fill the sketch slot.

DART-SQL [26] builds no index and applies no pre-processing. For value reference detection, it treats the entire NLQ as a single value reference and encodes it with GloVe [30]. For value mapping, it performs an exhaustive row-level search: each row embedding is the average of its cells' GloVe vectors, and cosine similarity to the NLQ embedding selects the top-N rows. No filtering is applied. As the mapping is performed at the row level, all values within the top-N selected rows are considered as value links, and they are appended to the LLM prompt.

OpenSearch-SQL [38] builds a dense index over embeddings of all string values. It takes the raw NLQ (no pre-processing) and uses an LLM to extract candidate value references. For value mapping, it queries the index with embeddings of these mentions and retrieves the top-K matches by cosine similarity. The filtering step then refines this set of candidates by retaining only those that exceed a similarity threshold. The final set of retained value links is appended to the prompt of the LLM used for SQL generation.

6 The VLD Benchmark

The commonly used BIRD [23] and Spider [40] benchmarks, have limited capacity to evaluate value linking. In Spider, only 21% of dev and test examples contain a value in the SQL, with exact matches found in the NLQ in 99% and 97% of cases, respectively. In BIRD, 87% of questions in its dev set require a value in the SQL, which is an improvement, yet 72% of these can still be found as an exact match in the NLQ. For the remaining non-exact matches, 40% belong to 'categorical' columns with fewer than five distinct values and 20% are date-related, involving different formats (e.g., '1996/10/21' vs '1996-10-21'). Given these limitations, a benchmark is needed to capture cases where NLQ value references deviate from exact database matches.

Group	Category	# Q	Example Transformation	Rej. Rate
Typographical	Typo-deletion	84	'nonparametric' → '**noparametric**'	0.80
	Typo-insertion	99	'South Palma' → '**South Palama**'	0.70
	Typo-substitution	147	'North Omer' → '**North Oner**'	0.60
	Typo-transposition	102	'Tumbleweed' → '**Tumbelweed**'	0.65
Formatting	Space Addition	147	'Autobiographer' → '**Auto biographer**'	0.95
	Space Removal	120	'Agent Zero' → '**AgentZero**'	0.80
	Punctuation Change	144	'hypothesis-testing' → '**hypothesis_testing**'	0.94
	Punctuation Removal	162	'Self-Learner' → '**Self Learner**'	0.70
Structural	Word to Symbol	138	'Night and Day' → '**Night & Day**'	0.75
	Word Removal	43	'Kipling West' → '**Kipling**'	0.85
	Word Addition	98	'Bachelor' → '**Bachelor level**'	0.90
	Word Order	124	'Port Chelsea' → '**Chelsea Port**'	0.80
	Singular/Plural	109	'Durability' → '**durabilities**'	0.70
Semantic	Abbreviation	165	'Advertisement' → '**Ad**'	0.93
	Clipping	99	'Advertisement' → '**Advert.**'	0.85
	Synonym	121	'female' → '**woman**'	0.93
	Paraphrase	63	'Hidden agenda' → '**Agenda is hidden**'	0.96
	Negated Antonym	55	'Disciplined' → '**not undisciplined**'	0.94
Total		**2,020**		**0.82**

Table 2. VLD-Bench taxonomy of value discrepancies, grouped by category, with number of queries and rejection rate per category.

We introduce **VLD-Bench**, the **V**alue **L**inking **D**iscrepancy **Bench**mark. The foundation of VLD-Bench is a comprehensive taxonomy of potential discrepancies that goes beyond simple synonyms and covers a wide range of linguistic and typographical variations that can occur in NLQs. Table 2 presents our 18 defined categories, each with a description and an example showing the change from the original value reference to the perturbed one.

The construction of VLD-Bench is a bottom-up procedure that starts from the generation of value discrepancies that can then be plugged into NLQ-SQL pairs. This is a deliberate choice against directly working on entire NLQs, and our motivation is based on two observations. First, LLMs are powerful in generating plausible and linguistically diverse value variations; however, when asked to rewrite entire questions, they often introduce hallucinations or nonsensical changes. In contrast, they are reliable at a more focused task: given a single value and a specific perturbation to be applied, they can accurately determine if the perturbation makes sense and what the output should be. For example, given the word 'cat', an LLM can return 'kitten' as a valid synonym, while for 'James Smith', it can determine that a synonym is not applicable. Second, we observed that the values in existing NLQ-SQL pairs are not always suitable for introducing every perturbation category. For instance, some values may not have any punctuation, making it impossible to perform punctuation removal. Fortunately, a rich resource of potential values exists just beneath our feet: the database content itself. Our approach combines these two observations. We first explore the database content to find values applicable for each perturbation category and generate their perturbations. Then, we inject these value-perturbation pairs into existing benchmark questions.

The generation process of VLD-Bench is described below:

(1) Query Curation. We gather queries with `WHERE` or `HAVING` clauses from BIRD dev and Spider dev/test sets (1327 queries).

(2) Filtering for Exact Matches. We further filter the above queries to only keep queries where the NLQ's value references exactly match database values, resulting in 1006 queries. This allows us to swap the values in the remaining queries with other values from the same column, helping us introduce more perturbations, while reducing the risk of creating low-quality examples.

(3) Value Selection. For each perturbation category, we randomly select up to 10,000 text values from columns in `WHERE` or `HAVING` clauses, creating a pool of candidate values to be perturbed.

(4) Value Perturbation. For each perturbation category, we utilized `Qwen2.5-32B-Instruct` to generate candidate discrepancies. We designed a specific prompt [3] for each of the 18 categories in our taxonomy, instructing the LLM to transform the original database value into a perturbed variant adhering to the target category or to generate a special token if the perturbation is not applicable. This creates an initial pool of LLM-generated value-perturbation pairs.

(5) Manual Curation. As the raw LLM output can contain hallucinations, we manually curated the pool using a *greedy curation strategy*. For each perturbation category, we grouped the candidate values generated by the LLM by column. Annotators then manually reviewed these candidates until they identified up to three distinct, high-quality examples per-column that could be replaced in any NLQ without altering its meaning. Once the target number of valid candidates was met for a given column, the remaining candidates were not evaluated. We present rejection rate per category in Table 2, calculated as the percentage of values that were rejected over the total values that were annotated for each category. For example, a rejection rate of 0.90 essentially means that for every value that was accepted into the benchmarks, the annotators rejected 9 values. Higher rejection rates (e.g., 0.96 for Paraphrase) indicate that the LLM often produced unrealistic or off-category variants, requiring substantial manual filtering. In contrast, lower rates (e.g., 0.60–0.70 for typos) suggest the LLM was more reliable for surface-level alterations. However, for complex categories like paraphrases, valid candidates were often scarce, making it impossible to find three examples for every column. To ensure the benchmark remained large enough, annotators compensated for these gaps by oversampling: they selected more than three examples from columns where valid candidates were plentiful to make up for the columns where they were rare.

(6) Instance Injection. For each value-perturbation pair, we find original NLQ-SQL pairs using that column, replace the value reference with the perturbed form, update the SQL with the unperturbed value, and retain only queries with non-empty SQL results to ensure reliable ground truth. This process results in 2054 queries covering all perturbation categories.

(7) Deduplication. Our value replacement procedure is applied to each value reference in isolation, without awareness of other values in the same NLQ. Consequently, this can generate queries with redundant logical conditions. For example, an NLQ might be altered to ask for "cyclists from 'GB' or from 'Great Britain'", which would correspond to a nonsensical SQL clause like `WHERE country = 'GB' OR country = 'GB'`. To eliminate these queries, we use an LLM to flag such cases for removal, followed by a manual review to ensure only these redundant queries are discarded.

Quality Assurance. To ensure consistency and reliability, the curation and validation process was conducted by two researchers specializing in database systems and natural language interfaces. We established two primary selection criteria: (1) *Category Adherence*, ensuring the perturbed value strictly matches the definition of the assigned category; and (2) *Linguistic Plausibility*, ensuring the perturbation reflects natural variations likely to be used by real users. We employed a cross-validation protocol where the dataset was split, and selections from one annotator were validated by the other. Disagreements were resolved through discussion, resulting in a 3% inter-annotator rejection rate during the validation phase.

[3]The prompts used for each category are available at https://github.com/athenarc/experimental-analysis-of-value-linking/tree/main/prompts.

The final VLD-Bench dataset consists of 2,020 queries, with 1,811 derived from the BIRD development set and 209 from the Spider development and test sets. Crucially, this constitutes a *parallel corpus*: for every perturbed query in VLD-Bench, there exists a corresponding "unperturbed" query in the source dataset with identical SQL logic, allowing for a fair comparison.

7 Evaluation

We evaluate modern text-to-SQL systems' robustness against realistic value discrepancies, addressing the following research questions:

RQ1: How robust is value linking to linguistic discrepancies? (Section 7.2) We evaluate the isolated value linking component of each system to measure its performance degradation when faced with the discrepancies introduced in VLD-Bench.

RQ2: How does value linking impact end-to-end performance? (Section 7.3) We measure the end-to-end execution accuracy of full text-to-SQL systems to quantify how failures in value linking propagate to the SQL generation.

RQ3: How do systems utilize value links and tolerate false positives? (Section 7.4) We assess how effectively systems incorporate correct value links and how their performance degrades when irrelevant links are introduced.

RQ4: What are the performance trade-offs of different detection and mapping methods? (Section 7.5) We systematically evaluate combinations of value reference detection and value mapping methods to analyze their accuracy and latency across discrepancy categories.

RQ5: How do value linking mechanisms scale with increasing database size? (Section 7.7) We evaluate the efficiency of different value linking designs by measuring indexing time, storage requirements, and query latency across databases of varying sizes.

Additionally, in Section 7.6, we apply our perturbation generation process to an independent in-house dataset built over the OpenAIRE Research Graph to validate that our findings generalize beyond Spider and BIRD, to completely unseen data.

7.1 Experimental Setup

We selected representative open-source systems from the BIRD leaderboard[4], with explicit value linking component: **OpenSearch-SQL** [38], **OmniSQL** [22], **CSC-SQL** [33], and **CHESS** [36]. To represent earlier but foundational approaches, we also include **BRIDGE** [24] and **ValueNet** [3], which employ exhaustive search strategies.

In addition, to assess how text-to-SQL systems perform without a dedicated value linking mechanism, we include **Contextual-SQL** [1], a text-to-SQL system relying on random value sampling, as well as three frontier LLMs: **Claude Sonnet 4.5**, **DeepSeek-V3.1**, and **GPT-5.2**. The latter are evaluated via direct prompting, where the model is provided with the NLQ and the database schema but lacks a value linking component. This allows us to assess how well modern LLMs can resolve value discrepancies through internal reasoning and general knowledge alone.

We utilize the provided fine-tuned models for CSC-SQL, OmniSQL, and Contextual-SQL, while for systems relying on a general-purpose LLM (CHESS and OpenSearch), we employ the open-source **Qwen2.5-Coder-32B-Instruct**[5]. For all embedding-based retrieval and similarity computations, we use the open-source **bge-m3**[6].

VLD-Bench queries are unseen by all systems: derived from dev/test sets while fine-tuned systems (OmniSQL, CSC-SQL, Contextual-SQL) used only training splits, and prompting-based systems

[4]https://bird-bench.github.io/
[5]https://huggingface.co/Qwen/Qwen2.5-Coder-32B-Instruct
[6]https://huggingface.co/BAAI/bge-m3

	Spider		BIRD		VLD-Bench		
System	**P**	**PR**	**P**	**PR**	**P**	**PR**	**Lat.**
OpenSearch	0.73	0.79	0.32	0.76	0.29	0.50	2.12
Omni/CSC-SQL	0.73	0.90	0.28	0.95	0.10	0.11	0.38
CHESS	0.69	0.79	0.16	0.75	0.13	0.43	4.08
ValueNet	0.38	0.90	0.11	0.86	0.08	0.53	20.32
BRIDGE	0.67	0.90	0.27	0.93	0.10	0.14	30.41

Table 3. Value linking component performance. 'P' stands for Precision, 'PR' for Perfect Recall, and Latency is measured in seconds.

(CHESS, OpenSearch-SQL) have no task-specific fine-tuning. We evaluate both the value linking components and the end-to-end text-to-SQL performance of these systems. For Contextual-SQL and the three frontier LLMs, which lack an explicit value linking module, we report only its end-to-end results.

We evaluate performance using three standard metrics. **Precision (P)** measures the fraction of retrieved value links that are correct, i.e., values required by the ground-truth SQL query. **Perfect Recall (PR)** is a strict binary metric equal to 1 if all required value links are correctly retrieved and 0 otherwise. We opt for this strict version of recall because omitting even one required value link would lead to an incorrect SQL query. Finally, **Execution Accuracy (EX)**, following standard evaluation protocols [23], quantifies end-to-end text-to-SQL performance as the percentage of predicted queries that yield the same results as the ground-truth query.

The experiments were conducted on a server with these specifications: **CPU**: Intel Xeon Gold 5318Y processor (2.10 GHz); **RAM**: 377 GB DDR4; **GPU**: 2× NVIDIA RTX A6000 (46 GB VRAM); **Software**: Ubuntu 22.04.3 LTS, Python 3.12, PyTorch 2.7.1, CUDA 12.2

7.2 Evaluation of Value Linking Component

To answer RQ1, we isolate the value linking component to evaluate robustness and efficiency, comparing VLD-Bench performance against an exact-match baseline. This baseline is constructed from the same subset of Spider and BIRD examples that VLD-Bench was derived from, where all value references in the NLQ are exact matches to their corresponding database entries. In this ideal setting, a system's task is merely to copy values verbatim into the SQL query. By contrasting performance on VLD-Bench against this baseline, we can precisely quantify the degradation caused by realistic value variations. Our analysis is twofold: first, we assess overall performance and latency to understand architectural trade-offs. Second, we provide a fine-grained breakdown of the results across our 18 discrepancy categories to pinpoint specific linguistic weaknesses and connect them to different architectural designs.

Table 3 presents the performance of the isolated components across the benchmarks, using the default thresholds defined in the original implementation of each system. As OmniSQL and CSC-SQL share an identical value linking component, we group their results under the label Omni/CSC-SQL. VLD-Bench discrepancies significantly degrade performance. Systems relying on exact token matches (OmniSQL, CSC-SQL, BRIDGE using BM25/LCS) experience 76-84% PR drops. This is because their algorithms are highly sensitive to any string variations. While the remaining systems appear more robust, they still suffer a significant PR drop of 26% to 37%. The sharp decline in P across systems reveals ineffective filtering of false positives when faced with linguistic ambiguity.

Table 3 shows exhaustive search strategies (ValueNet, BRIDGE) incur unacceptably high latencies (20.32-30.41s). OpenSearch and CHESS, which leverage LLM for value reference detection also introduce non-negligible overhead with latencies of 2.12 and 4.08 seconds. In contrast, OmniSQL and CSC-SQL, combining n-gram detection with BM25 index, achieve a low latency of 0.38 seconds.

Evaluating systems based on their default, hard-coded thresholds can be misleading, as these were experimentally tuned for different benchmarks. To assess core ranking capabilities fairly, we remove them and report perfect recall at rank k (PR@k).

Figure 2 shows that, on the baseline benchmarks, all systems achieve high PR, quickly surpassing 0.8 and plateauing close to 0.9 as k increases. In contrast, performance on VLD-Bench shows a severe degradation: for instance, OpenSearch-SQL and CHESS struggle to reach 0.7 PR, while BRIDGE plateaus below 0.4 even when considering the top 20 candidates. Interestingly, the performance ranking shifts on VLD-Bench: OmniSQL/CSC-SQL, the top performer on the baseline benchmarks, becomes one of the weakest on VLD-Bench. Conversely, OpenSearch-SQL and CHESS prove to be the most robust systems against value discrepancies.

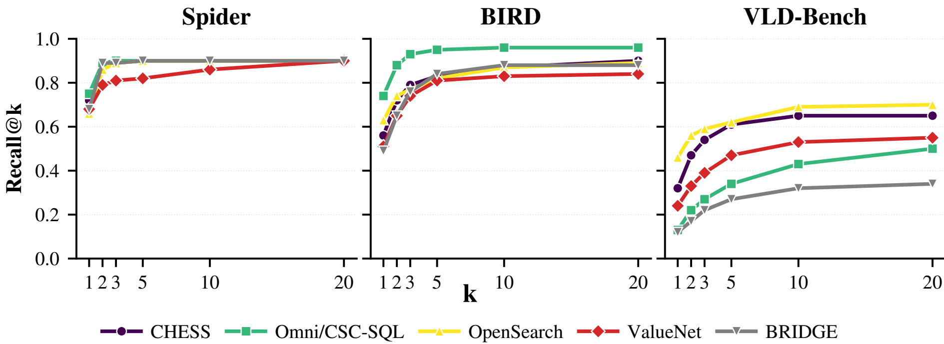

Fig. 2. Perfect recall @k across benchmarks

Figure 3 visualizes the relative drop in PR for each discrepancy category in VLD-Bench. This metric quantifies performance degradation, computed as $(1 - \text{PR}_{\text{VLD-Bench}}/\text{PR}_{\text{Baseline}})$. Consequently, a value of 1.0 (deep red) indicates complete failure (100% drop), while 0.0 (deep blue) indicates no degradation.

In the **Typographical** group, OmniSQL and BRIDGE show severe degradation due to BM25/LCS intolerance to character-level errors. Conversely, ValueNet remains almost completely unaffected, since its Damerau-Levenshtein metric is specifically designed to handle such discrepancies. This is because Damerau-Levenshtein assigns a low-cost penalty to the single-character edits that cause token-based (BM25) and subsequence-based (LCS) methods to fail. The similar performance of CHESS and OpenSearch, with PR drops from 0.16 to 0.60, is explained by their shared architectural bottleneck: a final filtering step based on embedding similarity. Thus, even when MinHashLSH correctly identifies a value link, it may be discarded by the semantic filter, explaining the similar PR drops in both systems.

Similar trends appear in the **Formatting** group. ValueNet shows resilience to single-character removals (Punctuation and Space Removal) but degrades on Space Addition and Punctuation Change. Meanwhile, the strictly lexical methods of OmniSQL and BRIDGE fail completely.

In the **Structural** group, systems relying solely on lexical similarity (OmniSQL, ValueNet, BRIDGE) are highly sensitive to most categories. In contrast, CHESS and OpenSearch, which employ a final semantic filtering step, are more resilient, with a performance drop below 17% in

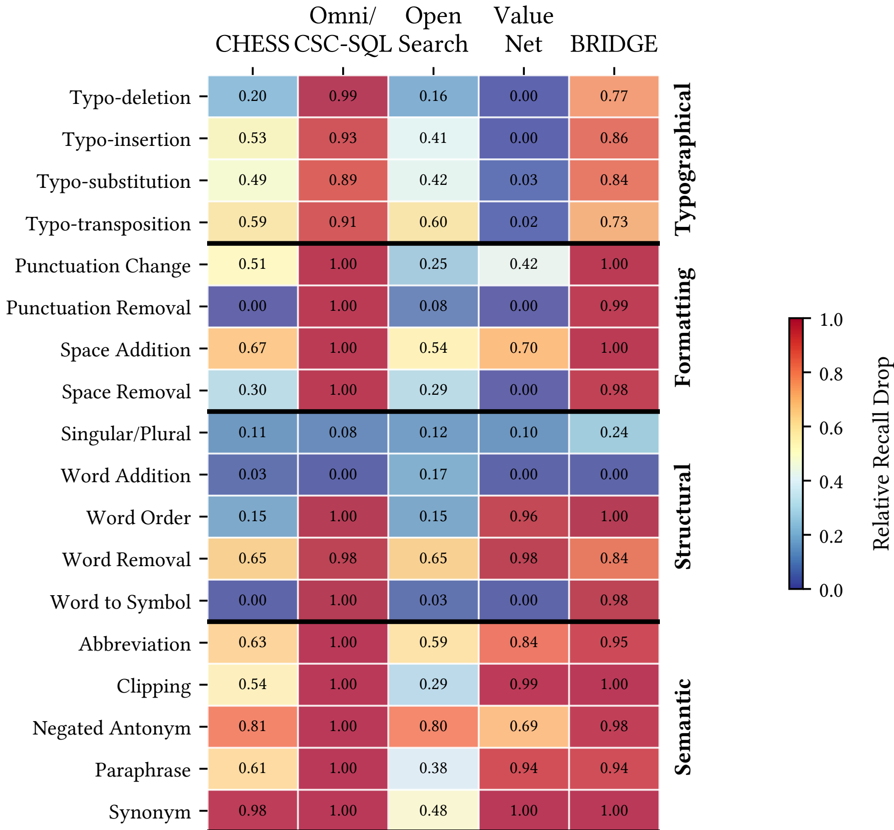

Fig. 3. Relative performance drop in perfect recall

four of the five categories. Their only severe degradation occurs with Word Removal, suggesting their embeddings are robust to word reordering and addition but not to removal.

Finally, the **Semantic** group poses the greatest challenge. Besides OpenSearch, which is designed for semantic similarity, all other systems exhibit a near-complete performance collapse, as their lexical methods are incapable of capturing meaning. However, even OpenSearch is not immune, with a non-negligible degradation ranging from 0.29 to 0.80. This confirms that accurately linking values based on semantic equivalence remains a major challenge.

7.3 Evaluation of the Text-to-SQL Task

To answer RQ2, we evaluate full text-to-SQL pipelines on both baseline and VLD-Bench to measure how value linking failures propagate to SQL generation. Table 4 presents the results across Spider, BIRD, and VLD-Bench. It is important to note that this table is not intended as a competitive leaderboard measuring absolute system-to-system performance, as results are confounded by differences in underlying LLM backbones and prompting strategies. Instead, the analysis should be interpreted row-wise: we focus on the *relative degradation* of each individual system when moving from baseline benchmarks to VLD-Bench. To ensure a fair comparison focused on modern architectures, we exclude the BRIDGE and ValueNet from this analysis. Their significantly lower baseline performance on benchmarks like BIRD would obscure the specific impact of value linking

discrepancies, making it difficult to isolate the degradation caused solely by VLD-Bench.

System	Spider	BIRD	VLD-Bench
OpenSearch-SQL	0.76	0.74	0.57
OmniSQL	0.88	0.77	0.25
CHESS	0.77	0.67	0.23
CSC-SQL	0.77	0.64	0.13
Contextual-SQL	0.84	0.73	0.36
Claude Sonnet 4.5	0.81	0.70	0.35
DeepSeek-V3.1	0.80	0.62	0.18
GPT-5.2	0.78	0.66	0.25

Table 4. Execution accuracy degradation on VLD-Bench compared to baseline benchmarks

The results confirm that VLD-Bench is substantially more challenging. All evaluated systems experience a sharp drop in accuracy when moving from BIRD and Spider to VLD-Bench, with performance degradation ranging from 17% for OpenSearch-SQL to a steep 64% for CSC-SQL. This performance degradation aligns with our component-level analysis: systems relying solely on lexical retrieval (OmniSQL and CSC-SQL) show the steepest accuracy decline. Their inability to handle linguistic variation at the component level directly impairs SQL generation. In contrast, OpenSearch-SQL, whose dense vector index provides greater semantic robustness, shows comparatively smaller performance drop, confirming the end-to-end benefits of a more resilient value linking mechanism.

We further observe that frontier LLMs exhibit similar fragility. While they achieve strong performance on the baselines, their accuracy collapses on VLD-Bench. DeepSeek-V3.1, for instance, drops from 0.80 on Spider to 0.18 on VLD-Bench. This confirms that implicit reasoning is insufficient for value linking: even the most advanced models struggle to perform value mapping latently when faced with the discrepancies introduced in our benchmark.

To further examine the latent value linking capabilities of frontier LLMs, we introduce latent value linking accuracy, measuring the proportion of generated SQL queries where the model correctly resolves the value discrepancy by using the correct database value. We compute this by parsing the generated SQL and comparing the inserted value against the ground-truth database entry, regardless of overall execution success. Before perturbation, 94% of queries used correct values; after perturbation, this drops to 32%. While these models could previously identify the correct value in nearly every query, only roughly 1 in 3 perturbed queries now include correct values—potentially due to random sampling of plausible values or successfully resolving certain discrepancy categories.

This evaluation provides a clear answer to RQ2: text-to-SQL systems, regardless of their architecture, are not robust to realistic value discrepancies. While all systems perform competitively on baseline benchmarks, their accuracy degrades severely on VLD-Bench—including frontier LLMs. This highlights that handling value discrepancies remains a critical unsolved challenge for text-to-SQL.

7.4 Ideal Value Links and Noise

To answer RQ3, we evaluate how systems utilize value links and tolerate false positives in two settings. First, we provide systems with oracle value links, representing an ideal high-PR, high-P scenario. Second, we introduce an increasing proportion of false positive value links to simulate a

high-PR, low-P scenario. To ensure our conclusions are not artifacts of a specific noise distribution, we employ two distinct noise injection protocols. (1) **Retrieval-based Noise (Hard Negatives):** We utilize the BM25 module from OmniSQL to retrieve candidate values from the *entire* database based on the NLQ. This provides high-ranking false candidates and represents a realistic challenging scenario where the system must distinguish between the correct value and lexically similar distractors. (2) **Same-Column Noise:** We randomly sample values from the *same column* as the ground truth value. By measuring execution accuracy on VLD-Bench across these conditions, we assess whether pursuing higher PR and P in value linking is warranted.

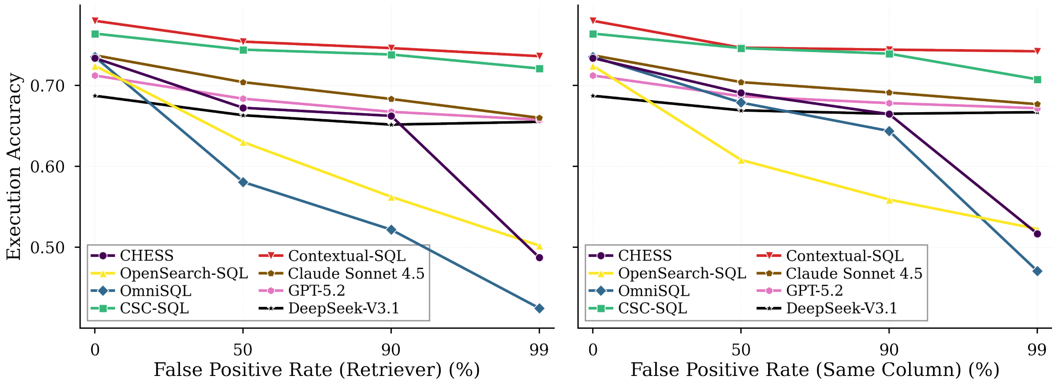

Fig. 4. Execution accuracy of text-to-SQL over different false positive rates

Focusing on the final text-to-SQL output, Figure 4 presents the execution accuracy of each system as the rate of false positives increases from an ideal 0% up to 99% under both noise protocols. First, we observe that providing correct value links boosts performance for all systems—a benefit that remains substantial even when a high volume of false positives is included. With ideal P, all systems achieve execution accuracy near their exact-match baseline performance (see Table 4). Contextual-SQL and CSC-SQL even surpass their baseline scores, indicating that correct value links help the model resolve other ambiguities in the SQL generation process. Second, comparing the two noise settings, we observe that the performance degradation trends remain consistent. The fact that the curves are nearly identical indicates that the systems' ability (or inability) to filter noise is an intrinsic architectural property, independent of whether the noise comes from lexically similar global matches or random column-constrained samples.

As the rate of false positives increases, the performance of all systems degrades, but their resilience to noise varies. The key distinction lies between single-pass generation systems and multi-stage systems that employ post-generation correction or selection. Contextual-SQL and CSC-SQL exhibit remarkable robustness, with their execution accuracy dropping by less than 5% even at a 99% false positive rate. Their multi-stage approach—generating a pool of candidate SQL queries followed by fine-tuned selection—effectively filters out SQL queries that do not use the correct value links. Similarly, CHESS and OpenSearch-SQL maintain reasonable performance until false positives reach 90%. This resilience can be attributed to their use of post-generation validation steps, which leverage additional calls to general-purpose, non-fine-tuned LLMs to assess the semantic correctness of the generated SQL. In contrast, OmniSQL's single-pass generation without revision or filtering leads to the steepest degradation, with accuracy dropping from 0.74 to 0.42.

A similar robustness characterizes the frontier models. While these models struggled on VLD-Bench, providing oracle links triggers an immediate recovery, boosting execution accuracy to over

Val. Ref. Det.	Value Mapping	Typos		Format		Structural		Semantic		Avg. Lat. (s)
		P	PR	P	PR	P	PR	P	PR	
Full NLQ	Exhaustive	0.00	0.00	0.00	0.00	0.00	0.00	0.00	0.00	9.36
	BM25 Index	0.11	0.07	0.07	0.01	0.18	0.35	0.07	0.01	0.02
	MinHashLSH Index	0.00	0.00	0.25	0.01	0.25	0.01	0.00	0.00	0.03
	Dense Index	0.00	0.00	0.25	0.01	0.25	0.01	0.00	0.00	0.16
n-grams	Exhaustive	–	–	–	–	–	–	–	–	–
	BM25 Index	0.10	0.07	0.06	0.01	0.17	0.37	0.06	0.01	0.38
	MinHashLSH Index	0.13	0.42	0.13	0.49	0.20	0.71	0.10	0.21	2.94
	Dense Index	0.08	0.50	0.09	0.58	0.11	0.77	0.08	0.46	13.47
NER	Exhaustive	0.09	0.91	0.07	0.61	0.08	0.64	0.02	0.13	20.28
	BM25 Index	0.14	0.07	0.08	0.01	0.19	0.36	0.00	0.00	0.05
	MinHashLSH Index	0.15	0.45	0.15	0.47	0.24	0.73	0.11	0.25	1.34
	Dense Index	0.20	0.51	0.18	0.49	0.23	0.76	0.15	0.41	1.24
LLM	Exhaustive	0.13	0.91	0.12	0.67	0.10	0.55	0.04	0.14	17.75
	BM25 Index	0.11	0.07	0.07	0.01	0.17	0.37	0.07	0.01	1.64
	MinHashLSH Index	0.16	0.49	0.15	0.44	0.22	0.63	0.10	0.24	4.08
	Dense Index	0.19	0.50	0.20	0.47	0.23	0.67	0.15	0.41	2.12

Table 5. All combinations of Value Reference Detection and Value Search Techniques per category of VLD-Bench. Measuring Precision, Perfect Recall, and Average Latency in seconds.

70%. Furthermore, their performance remains mostly stable under noise, maintaining >65% accuracy even with 99% false positives.

This clearly answers RQ3: solving value linking can address linguistic discrepancies in VLD-Bench. Providing perfect value links consistently boosts execution accuracy, underscoring the need for robust value linking. Moreover, while false positives affect systems differently, the most resilient architectures perform nearly as well with high-PR, imprecise links as with perfectly precise ones.

7.5 Performance Impact of Detection and Mapping Methods

To answer RQ4, we systematically evaluate all combinations of value reference detection and value mapping methods on VLD-Bench. By breaking down the results by discrepancy category, this fine-grained analysis uncovers the relationships between specific strategies and their effectiveness at handling different linguistic variations, revealing the strengths and weaknesses of each choice. Table 5 presents the P, PR, and average latency for each combination, For the **Exhaustive Search** method, we adopt the implementation from ValueNet, as our preliminary experiments showed it to be faster and more effective than the approach used in BRIDGE.

Value Reference Detection. The detection method sets the upper bound for value linking PR—missed references in the NLQ cannot be recovered by the value mapping. Its P also affects computational cost by limiting the number of candidate values passed to the mapping stage. Using the **Full NLQ** as a single reference results in near-zero PR for almost all configurations, as mapping methods are not designed to locate a value span within a long, noisy query. At the other extreme, the brute-force **n-grams** method ensures that value references are not missed, achieving the highest PR in the Structural (0.77) and Semantic (0.46) groups when paired with a Dense Index, but its low P (0.08-0.13) introduces significant noise and computational overhead. This method is computationally prohibitive when paired with an exhaustive search, with each query requiring over a minute to process; therefore, these results are omitted from the table. The overhead is also evident when paired with a Dense Index, where the high volume of candidates leads to an impractical latency of 13.47s. In contrast, targeted methods like **NER** and **LLM** provide a much better balance.

By precisely identifying value references, they enable the use of powerful but computationally intensive mapping methods at a manageable latency. For example, using NER instead of n-grams with a Dense Index reduces latency more than tenfold, from 13.47s to 1.24s, while maintaining high PR. Both NER- and LLM-based methods show similar performance, making them the most effective strategies for value reference detection.

Value Mapping. The value mapping method determines a system's robustness to different types of discrepancies once a value reference is identified. **Exhaustive Search** achieves up to 0.91 PR on typos with NER/LLMs but has prohibitive latency (>17s), unsuitable for interactive systems. While **BM25 Index** is the fastest method (0.02s-1.64s) but performs poorly, with PR rarely exceeding 0.07 on typographical and semantic discrepancies. **MinHashLSH Index** offers a middle ground, providing moderate robustness to typographical and structural variations. By approximating Jaccard similarity, it tolerates character-level noise better than BM25, achieving 0.73 PR on structural changes at 1.34s latency. Its weakness is semantic discrepancies (PR $\leq$ 0.25). Dense Index is the only strategy with meaningful semantic performance (PR up to 0.46). Its effectiveness comes with variable latency that is highly dependent on the number of candidates from the detection stage, ranging from 1.24s with NER to 13.47s with n-grams.

Discrepancy Groups. Typographical and Formatting discrepancies are best handled by a precise value reference detector paired with a mapping method that allows for approximate matching. While exhaustive search is most effective (0.91 PR), the practical solution is a MinHashLSH or Dense Index, which maintains moderate PR (0.45-0.51) at an acceptable latency. The **Structural** group highlights the gap between sparse and dense value mapping methods. PR jumps from a maximum of 0.37 with a BM25 index to over 0.76 with a Dense Index. While the n-grams + Dense Index combination achieves the highest PR here (0.77), the NER + Dense Index pairing is nearly as effective (0.76) and more than ten times faster. The **Semantic** group is the most challenging and represents the primary source of failure. It is the only category where the mapping method must be semantic. The combination of a targeted detector (**NER** or **LLM**) with a **Dense Index** is the most effective strategy, achieving a 0.41 PR. Although the brute-force n-grams approach reaches a slightly higher PR (0.46), its prohibitive latency makes it unviable. This shows that even the best-performing practical combinations fail on nearly 60% of semantic discrepancies, making semantic variations the core unsolved problem in value linking.

In summary: BM25 offers the lowest latency but sacrifices robustness; MinHashLSH (1-4s) handles surface-level noise, but not semantic shifts; semantic robustness requires Dense Index with a precise detector (NER/LLM) for practical latency.

7.6 Generalization to Unseen Schemas

While we have ensured that the specific queries in VLD-Bench are unseen by the evaluated systems, Spider and BIRD are ubiquitous, well-established benchmarks. Given their prominence, it is possible that models—particularly frontier LLMs—may have encountered them during pre-training. To ensure that our findings are universal and not tied to the specific characteristics of these benchmarks, we conducted an additional evaluation on a completely independent, real-world dataset.

We applied our perturbation pipeline to an in-house dataset of 227 queries over the **OpenAIRE Research Graph**, yielding **843 perturbed NLQ-SQL** pairs across the 18 discrepancy categories.

	OpenAIRE-Original			OpenAIRE-Perturbed		
System	**P**	**PR**	**EX**	**P**	**PR**	**EX**
OpenSearch-SQL	0.13	0.67	0.34	0.11	0.47	0.18
OmniSQL	0.15	0.84	0.30	0.06	0.18	0.20
CHESS	0.06	0.67	0.32	0.04	0.47	0.16
CSC-SQL	0.06	0.67	0.34	0.04	0.47	0.24
BRIDGE	0.23	0.83	—	0.08	0.27	—
ValueNet	0.09	0.77	—	0.07	0.44	—
Contextual-SQL	—	—	0.26	—	—	0.10
Claude Sonnet 4.5	—	—	0.26	—	—	0.13
DeepSeek-V3.1	—	—	0.24	—	—	0.11
GPT-5.2	—	—	0.27	—	—	0.12

Table 6. Generalization results on the OpenAIRE hold-out dataset. PR and P for value linking component; EX for SQL Execution Accuracy.

Table 6 results support VLD-Bench trends. Lexical approaches show fragility: **OmniSQL** (BM25) PR collapses from 0.84 to 0.18; OpenSearch-SQL (dense retrieval) shows smaller drop **(0.67 to 0.47)**.

This degradation propagates directly to end-to-end execution accuracy (29-50% drops). Frontier LLMs show similar fragility (perturbed accuracy: 0.11-0.13), well below multi-component systems, underscoring the need for dedicated text-to-SQL components rather than sole reliance on out-of-the-box prompting. These findings confirm that sensitivity to value discrepancies persists across real-world schemas, not just Spider and BIRD.

7.7 Scalability Analysis

To answer RQ5, we move beyond the limited size of Spider and BIRD, which lack the volume required to expose the operational limitations of indexing strategies. We exclude exhaustive search approaches, as they do not scale and are not employed by any modern system. We utilize the OpenAIRE database introduced in Section 7.6, which contains approximately 350GB of data. To simulate realistic growth, we sampled an increasing number of distinct values to be ingested by the value linking components, ranging from 10K to 50M entries; for reference, the mean number of distinct values in the Spider and BIRD databases is 50K, with a maximum of 1.2M. We evaluate the indexing time, storage size, and query latency of different systems (Figure 5). While indexing is performed offline, its cost directly impacts storage requirements, and excessively slow indexing times can render a system impractical to deploy. OmniSQL's sparse index remains lightweight, reaching only 6GB in size and taking about 2.8 hours to build at the largest scale. On the other hand, CHESS takes over 19 hours to build a 342GB index, while OpenSearch-SQL takes about 11 hours to build a 195GB index.

When processing queries, the trade-off between speed and capability becomes clear. OmniSQL maintains a low latency of 0.59 seconds even at the largest scale, confirming that sparse retrieval is the only practical option for large databases. On the other hand, CHESS and OpenSearch-SQL become significantly slower as the data volume grows, reaching latencies of 43 seconds and 36 seconds, respectively, which is too slow for real-time interaction. This highlights a key limitation: while dense and MinHash indexes are better at handling discrepancies, they are too slow for large-scale use.

8 Implications for Real-World Deployment

Our analysis highlights a gap between text-to-SQL performance and industrial deployment requirements. To bridge this gap, we propose prescriptive guidelines for system architects.

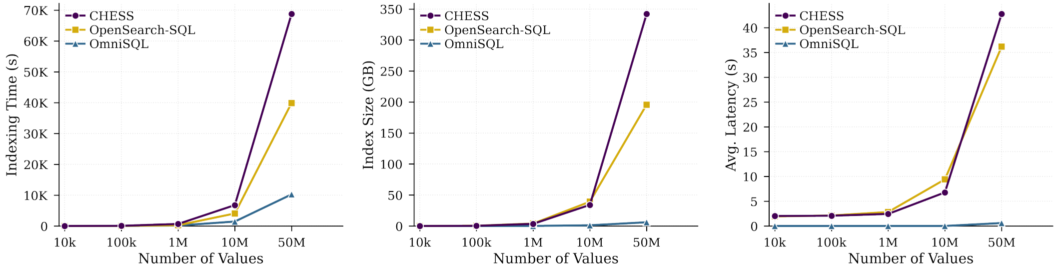

Fig. 5. Indexing time, storage size, and query latency across increasing database scales.

8.1 The "Data Blindness" Bottleneck

Academic benchmarks often rely on the assumption that users possess perfect knowledge of database content. In enterprise reality, this assumption fails, creating a significant bottleneck. Consider a data analyst asking, *"Show me sales for Dyson vacuums in NY."* If the database stores 'Dyson V15 Detect' and 'New York, NY', a standard text-to-SQL system suffers from *data blindness*. Without explicit grounding, the model generates a WHERE clause targeting the literal NLQ string, resulting in empty result sets.

Our evaluation confirms even capable systems with dedicated value linking experience severe accuracy drops on VLD-Bench. It is thus clear that value linking is not merely an optimization; it is an essential grounding step for system reliability, especially when facing real-world scenarios with real users. VLD-Bench exposes pain points that are overlooked by common benchmarks and provides a methodology to diagnose these failures. By measuring performance across specific variation categories, it allows designers to identify exactly which linguistic discrepancies (e.g., semantic or typos) lead to system collapse. Finally, designers can also reuse VLD-Bench's data augmentation pipeline to introduce value discrepancies in their own benchmarks and test how well their solutions perform under these conditions.

8.2 Lessons Learned and Key Takeaways

Using the results of our experimental study, we can derive multiple takeaways that can pave the path for future research on both text-to-SQL and Value Linking. First, it is evident that value discrepancies constitute a key factor in failure, leading to execution accuracy drops of up to 64%. However, our results offer a clear path forward. In an ideal scenario with oracle value links, system accuracy is fully restored. Crucially, multi-stage architectures like Contextual-SQL and CSC-SQL remain robust even at a 99% false positive rate. As such, the primary goal should be to maximize recall; a noisy set of candidate links is significantly better than missing the correct link entirely.

Our taxonomy deconstructs value linking into its discrete stages, revealing each component's role, necessity and performance characteristics. This enables architects to selectively combine and build upon existing solutions, composing more robust systems. Fast, lexical-based methods (e.g., BM25) are sufficient for surface-level robustness against typographical or formatting changes. However, they are brittle against semantic discrepancies. For true semantic understanding, the only viable path is a precise detector (NER or LLM-based) paired with a dense index.

Finally, while dense and MinHash indexes handle discrepancies better, their indexing time and storage footprint can be prohibitive. Given the differing capabilities of these methods across discrepancy types, future hybrid approaches could combine multiple matching strategies, though at increased implementation complexity. It is evident that future work must look for a way to

balance the need for semantic recall with the operational costs of maintaining large-scale vector embeddings.

9 Conclusion

This work presented the first systematic study of value linking in text-to-SQL systems, introducing a comprehensive taxonomy, a new benchmark (VLD-Bench), a comparative evaluation of state-of-the-art approaches, and insights on best practices. Given the importance of text-to-SQL in research and industry, and the demonstrated impact of value linking on its performance, greater focus from the data community is warranted. VLD-Bench along with experiments on multiple SOTA systems have shown a clear pain point so far ignored due to the lack of realistic value discrepancies in common benchmarks. However, these types of discrepancies are very common in real-world applications, where users are not familiar with the DB contents, and systems must be prepared to handle them. Our survey, taxonomy, and method-level analysis clarify the design space, compare available tools, and offer case-by-case recommendations with noted pitfalls. Finally, despite the trend toward relying on LLMs, value linking is not an LLM-only problem; progress will require stronger retrieval mechanisms that robustly handle diverse value discrepancies efficiently and effectively. Additionally, while we exclude numerical and date values, as their handling falls outside retrieval-based value linking, they remain crucial for real-world systems, and we encourage future work and benchmarks targeting numerical and temporal reasoning in text-to-SQL.

Acknowledgments

This work has been partially supported by DataGEMS, funded by the European Union's Horizon Europe Research and Innovation programme, under grant agreement No 101188416. This work has been partially supported by project MIS 5154714 of the National Recovery and Resilience Plan Greece 2.0 funded by the EuropeanUnion under the NextGenerationEU Program.

References

[1] Sheshansh Agrawal and Thien Nguyen. 2025. Open-Sourcing the Best Local Text-to-SQL System. https://contextual.ai/blog/open-sourcing-the-best-local-text-to-sql-system/

[2] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. (2020). arXiv:2005.14165 [cs.CL]

[3] Ursin Brunner and Kurt Stockinger. 2021. ValueNet: A Natural Language-to-SQL System that Learns from Database Information. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*. 2177–2182. doi:10.1109/ICDE51399.2021.00220

[4] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness. arXiv:2301.08881

[5] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S. Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the Twentieth Annual Symposium on Computational Geometry* (Brooklyn, New York, USA) *(SCG '04)*. Association for Computing Machinery, New York, NY, USA, 253–262. doi:10.1145/997817.997857

[6] Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-Grounded Pretraining for Text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tur, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, Online, 1337–1350. doi:10.18653/v1/2021.naacl-main.105

[7] Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The Faiss library. (2024). arXiv:2401.08281 [cs.LG]

[8] Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. 2023. CatSQL: Towards Real World Natural Language to SQL Applications. *Proc. VLDB Endow.* 16, 6 (Feb. 2023), 1534–1547. doi:10.14778/3583140.3583165

[9] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards Robustness of Text-to-SQL Models against Synonym Substitution. 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)*, Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli (Eds.). Association for Computational Linguistics, Online, 2505–2515. doi:10.18653/v1/2021.acl-long.195

[10] Yujian Gan, Xinyun Chen, and Matthew Purver. 2021. Exploring Underexplored Limitations of Cross-Domain Text-to-SQL Generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.). Association for Computational Linguistics, Online and Punta Cana, Dominican Republic, 8926–8931. doi:10.18653/v1/2021.emnlp-main.702

[11] Yujian Gan, Xinyun Chen, and Matthew Purver. 2023. Re-appraising the Schema Linking for Text-to-SQL. In *Findings of the Association for Computational Linguistics: ACL 2023*, Anna Rogers, Jordan Boyd-Graber, and Naoaki Okazaki (Eds.). Association for Computational Linguistics, Toronto, Canada, 835–852. doi:10.18653/v1/2023.findings-acl.53

[12] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, Jinyang Gao, Liyu Mou, and Yu Li. 2024. XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. *arXiv preprint arXiv:2411.08599* (2024). https://arxiv.org/abs/2411.08599

[13] Michael Glass, Mustafa Eyceoz, Dharmashankar Subramanian, Gaetano Rossiello, Long Vu, and Alfio Gliozzo. 2025. Extractive Schema Linking for Text-to-SQL. (Jan. 2025). arXiv:2501.17174 [cs.DB]

[14] Gemini Team Google. 2023. Gemini: A Family of Highly Capable Multimodal Models. arXiv:arXiv:2312.11805

[15] Zihui Gu, Ju Fan, Nan Tang, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Sam Madden, and Xiaoyong Du. 2023. Interleaving Pre-Trained Language Models and Large Language Models for Zero-Shot NL2SQL Generation. arXiv:2306.08891 [cs.CL]

[16] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. 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*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4524–4535. doi:10.18653/v1/P19-1444

[17] Haoyang Li and Jing Zhang and Hanbing Liu and Ju Fan and Xiaokang Zhang and Jun Zhu and Renjie Wei and Hongyan Pan and Cuiping Li and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data* 2, 3, Article 127 (May 2024), 28 pages. doi:10.1145/3654930

[18] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:arXiv:2406.08426

[19] Jeff Johnson, Matthijs Douze, and Hervé Jégou. 2017. Billion-scale similarity search with GPUs. arXiv:arXiv:1702.08734

[20] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-SQL. *VLDB J.* 32, 4 (2023), 905–936. doi:10.1007/S00778-022-00776-8

[21] George Katsogiannis-Meimarakis, Katsiaryna Mirylenka, Paolo Scotton, Francesco Fusco, and Abdel Labbi. 2026. In-depth Analysis of LLM-based Schema Linking.. In *EDBT*. 117–130.

[22] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale. arXiv:arXiv:2503.02240

[23] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C C Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as A database interface? A BIg bench for large-scale database grounded text-to-SQLs. (May 2023). arXiv:2305.03111 [cs.CL]

[24] Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging Textual and Tabular Data for Cross-Domain Text-to-SQL Semantic Parsing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 4870–4888. doi:10.18653/v1/2020.findings-emnlp.438

[25] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? Text-to-SQL in the age of well-reasoned language models. (Aug. 2024). arXiv:2408.07702 [cs.CL]

[26] Wenxin Mao, Ruiqi Wang, Jiyu Guo, Jichuan Zeng, Cuiyun Gao, Peiyi Han, and Chuanyi Liu. 2024. Enhancing Text-to-SQL Parsing through Question Rewriting and Execution-Guided Refinement. In *Findings of the Association for Computational Linguistics: ACL 2024*, Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.). Association for Computational Linguistics, Bangkok, Thailand, 2009–2024. doi:10.18653/v1/2024.findings-acl.120

[27] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. arXiv:arXiv:1310.4546

[28] Anna Mitsopoulou and Georgia Koutrika. 2025. Analysis of Text-to-SQL Benchmarks: Limitations, Challenges and Opportunities. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, Alkis Simitsis, Bettina Kemme, Anna Queralt, Oscar Romero, and Petar Jovanovic (Eds.).

OpenProceedings.org, 199–212. doi:10.48786/EDBT.2025.16

[29] Ali Mohammadjafari, Anthony S. Maida, and Raju Gottumukkala. 2024. From Natural Language to SQL: Review of LLM-based Text-to-SQL Systems. arXiv:arXiv:2410.01066

[30] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Alessandro Moschitti, Bo Pang, and Walter Daelemans (Eds.). Association for Computational Linguistics, Doha, Qatar, 1532–1543. doi:10.3115/v1/D14-1162

[31] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O. Arik. 2024. CHASE-SQL: Multi-Path Reasoning and Preference Optimized Candidate Selection in Text-to-SQL. arXiv:arXiv:2410.01943

[32] Stephen Robertson and Hugo Zaragoza. 2009. The Probabilistic Relevance Framework: BM25 and Beyond. *Found. Trends Inf. Retr.* 3, 4 (April 2009), 333–389. doi:10.1561/1500000019

[33] Lei Sheng and Shuai-Shuai Xu. 2025. CSC-SQL: Corrective Self-Consistency in text-to-SQL via reinforcement learning. (June 2025). arXiv:2505.13271 [cs.CL]

[34] Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2025. A Survey on Employing Large Language Models for Text-to-SQL Tasks. *ACM Comput. Surv.* 58, 2, Article 54 (Sept. 2025), 37 pages. doi:10.1145/3737873

[35] Robyn Speer and Catherine Havasi. 2012. Representing General Relational Knowledge in ConceptNet 5. In *Proceedings of the Eighth International Conference on Language Resources and Evaluation (LREC'12)*, Nicoletta Calzolari, Khalid Choukri, Thierry Declerck, Mehmet Uğur Doğan, Bente Maegaard, Joseph Mariani, Asuncion Moreno, Jan Odijk, and Stelios Piperidis (Eds.). European Language Resources Association (ELRA), Istanbul, Turkey, 3679–3686. https://aclanthology.org/L12-1639/

[36] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual harnessing for efficient SQL synthesis. (May 2024). arXiv:2405.16755 [cs.LG]

[37] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. 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*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault (Eds.). Association for Computational Linguistics, Online, 7567–7578. doi:10.18653/v1/2020.acl-main.677

[38] Xiangjin Xie, Guangwei Xu, Lingyan Zhao, and Ruijie Guo. 2025. OpenSearch-SQL: Enhancing Text-to-SQL with Dynamic Few-shot and Consistency Alignment. *Proc. ACM Manag. Data* 3, 3, Article 194 (June 2025), 24 pages. doi:10.1145/3725331

[39] Wenbo Xu, Liang Yan, Peiyi Han, Haifeng Zhu, Chuanyi Liu, Shaoming Duan, Cuiyun Gao, and Yingwei Liang. 2024. TCSR-SQL: Towards Table Content-aware Text-to-SQL with Self-retrieval. arXiv:arXiv:2407.01183

[40] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. 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*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. doi:10.18653/v1/D18-1425

Received October 2025; revised January 2026; accepted February 2026