# SQUiD: Synthesizing Relational Databases from Unstructured Text

**Mushtari Sadia** **Zhenning Yang** **Yunming Xiao**
**Ang Chen** **Amrita Roy Chowdhury**
University of Michigan, Ann Arbor
{mushtari, znyang, yunmingx, chenang, aroyc}@umich.edu

## Abstract

Relational databases are central to modern data management, yet most data exists in unstructured forms like text documents. To bridge this gap, we leverage large language models (LLMs) to automatically synthesize a relational database by generating its schema and populating its tables from raw text. We introduce SQUiD, a novel neurosymbolic framework that decomposes this task into four stages, each with specialized techniques. Our experiments show that SQUiD consistently outperforms baselines across diverse datasets. Our code and datasets are publicly available at: https://github.com/Mushtari-Sadia/SQUiD.

## 1 Introduction

Relational databases serve as the foundation for data management, supported by decades of mature infrastructure development and a wide array of sophisticated analytical tools. However, much of today's data exists as raw, unstructured text – such as academic articles, medical records, and business reports (Harbert, n.d.). This unstructured data cannot be directly analyzed using conventional database tools, which rely on structured, relational inputs. Bridging this gap remains a long-standing goal of the data management community (Mansuri and Sarawagi, 2006; Smith et al., 2022a; Chu et al., 2007; Yafooz et al., 2013; Michelson and Knoblock, 2008; Murthy et al., 2012; Jain et al., 2007), with a key challenge being the conversion of unstructured text into queryable, structured formats compatible with existing relational database infrastructure.

Large language models (LLMs) presents a unique opportunity to *automate* this conversion, owing to their growing capability to understand natural language and perform complex information extraction tasks. Prior work in this space can be broadly categorized into two areas. The first focuses on generating summarizing structures

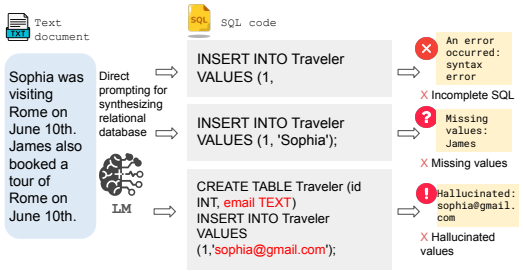


Figure 1: Challenges of synthesizing relational DB from text

from text, such as tables (Deng et al., 2024; Wu et al., 2022; Sundar et al., 2024; Li et al., 2023; Arora et al., 2023) and mind maps (Jain et al., 2024)—but these non-relational representations are often tailored for specific downstream applications (Shavarani and Sarkar, 2025; Sui et al., 2024), and lack the expressiveness and semantics of relational databases. The second category manipulates a *pre-defined* and fully populated relational database—e.g., Text-to-SQL (Hong et al., 2024) approaches generate executable SQL queries from text over given schemas, while a recent work can update existing relational databases using text input (Jiao et al., 2024). However, a key challenge of managing unstructured text is precisely that such a pre-defined database often does not exist.

In this paper, we pursue a more ambitious goal – *synthesizing a relational database from unstructured text from scratch*—a task that we call Text2R. The Text2R task presents several unique challenges. First, a relational schema consists of multiple interrelated tables that capture complex entity-relationship semantics, and it must also preserve syntactic integrity, such as satisfying primary/foreign key constraints. Second, database records must be correctly identified and populated across tables. This involves ensuring value consistency – e.g., the same entity must be consistently represented in all relevant tables. Third, the actual database creation requires valid and executable SQL statements, adding another layer of complexity. Naïve approaches, such as directly prompting

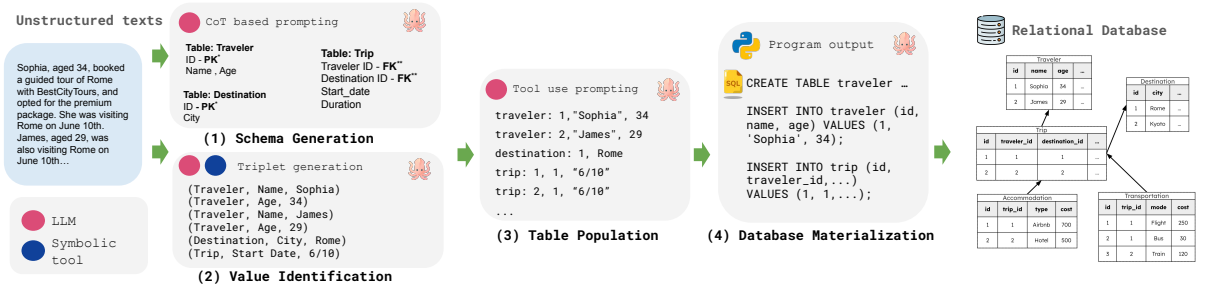


Figure 2: Overview of SQUiD. **(1) Schema Generation** constructs a relational schema that defines the tables, columns, and their relationships, from the entities in the text. **(2) Value Identification** extracts relevant values (e.g., names, dates) from the text. These values are then organized during **(3) Table Population** by aligning them with the generated schema to form tuples. **(4) Database Materialization** programmatically translates the output into SQL statements, producing the final relational database.

LLMs to synthesize databases, leads to diverse errors, including missing or hallucinated values, and SQL syntax issues (Fig. 1).

To address these challenges, we propose SQUiD[1], a neurosymbolic framework for the Text2R task. Our key idea is to decompose the task into multiple modular stages in a principled manner—breaking the problem into manageable sub-tasks. This allows each stage to leverage specialized techniques, such as symbolic information extraction and LLM-assisted tool use, for improved performance. Via task breakdown, some stages can also be executed programmatically, enhancing both accuracy and consistency. Additionally, each stage incorporates best practices from relational database literature to guide prompt design.

SQUiD consists of four stages, which generalize across text from diverse domains. The *schema generation* stage uses LLMs to infer a relational schema from the input text, guided by carefully designed prompts that incorporate best practices to identify entities and relationships. In the *value identification* stage, intermediate representations in the form of triplets are extracted using both symbolic tools and LLMs. These triplets break down complex sentences into granular units, improving coverage of the extracted values. Next, the *table population* stage aligns these triplets with the generated schema to form schema-consistent tuples. Finally, instead of generating SQL directly via LLMs—which can be token-intensive—our *database materialization* stage programmatically translates the structured outputs into valid SQL statements, ensuring syntactic correctness and structural fidelity. The resulting SQL is then executed to instantiate the final database. We make the following contributions:

- We define a new task – synthesizing relational databases from unstructured text, or Text2R. This marks a clear departure from prior work, which focuses on downstream relational tasks (e.g., Text2SQL), assuming a pre-existing database.
- We propose SQUiD, a novel neurosymbolic framework for Text2R, based on a four-stage decomposition. Each stage leverages custom techniques tailored to its specific subtask.
- We establish an automated benchmark methodology for Text2R. We also define a suite of evaluation metrics to assess schema and tuple quality along both semantic and syntactic dimensions.
- We conduct extensive experiments across diverse text domains and show that SQUiD consistently outperforms direct prompting baselines.

## 2 The Text2R Task

We begin by defining this new task of relational database synthesis, or Text2R. Given an unstructured document $D$ of natural language text, the goal is to produce a set of SQL statements $S$: (1) `CREATE TABLE` statements which define the schema $\mathcal{R}$, specifying the structure of the database in terms of tables and columns; and (2) `INSERT` statements which populate the relations with data extracted from the text in $D$. The schema $\mathcal{R}$ consists of a set of tables $\mathbf{T} = \{T_1, T_2, \ldots, T_n\}$ where each $T_i$ has a set of columns $\mathbf{C}_i = \{C_{i,1}, C_{i,2}, \ldots, C_{i,k_i}\}$. Each table corresponds to an entity type, and the tables are inter-related, organizing the extracted tuples from the text into a database. A tuple $t$ for table $T_i$ is represented as: $t = \langle v_1, v_2, \ldots, v_{k_i} \rangle$ where $v_j$ is the value corresponding to column $C_{ij} \in T_i$. Each tuple represents a unique *instance* of the entity described by $T_i$. Fig 3 illustrates the differences between Text2R and other tasks.

## 3 SQUiD Framework

SQUiD decomposes the Text2R task into four modular stages that mirror the typical database construction process. First, a relational database schema is designed by identifying the domain's entities and

[1]SQUiD - **SQ**L on **U**nstructured **D**ata

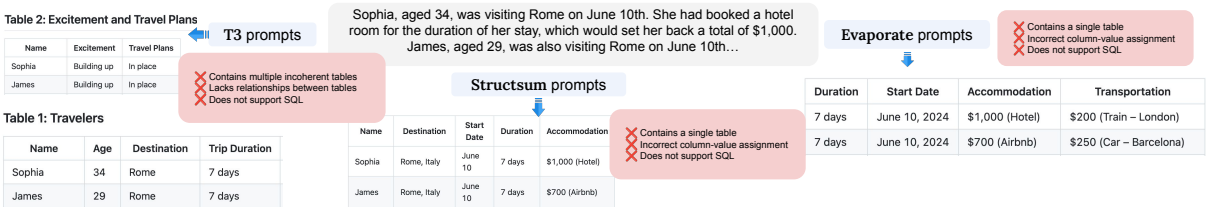


Figure 3: Closest related works—T3(Deng et al., 2024), STRUCTSUM(Jain et al., 2024), and EVAPORATE(Arora et al., 2023)—when applied to our example dataset, either produced a single table with incorrect column-value assignments or multiple disconnected, irrelevant tables. In contrast, as shown in Fig.2, SQUiD correctly generates all five tables corresponding to the entities (*Traveler*, *Trip*, *Accommodation*, *Transportation* and *Destination*) along with their proper relationships.

relationships—this is the **schema generation** stage. Next, SQUiD extracts all the relevant values from the text (**value identification**), which are then used to construct tuples (**table population**). Finally, the generated schema and tuples are translated into valid SQL statements during the **database materialization** stage. We describe these stages below, using the following text shown in Fig. 2 as a running example: *"Sophia booked a guided tour of Rome with BestCityTours, and opted for the premium package. She was visiting Rome on June 10th. James, aged 29, was also visiting Rome on June 10th."*

### 3.1 Schema Generation

**Challenge.** The complexity of schema generation is both semantic and syntactic. Semantically, the schema must accurately capture the entity-relationship structure that reflects the underlying data. Syntactically, a valid schema must comply with the integrity constraints defined by the established principles of relational databases. Simply prompting LLMs to generate a schema without explicitly articulating the necessary relational database constraints can result in structurally invalid outputs, as illustrated in Fig. 4.

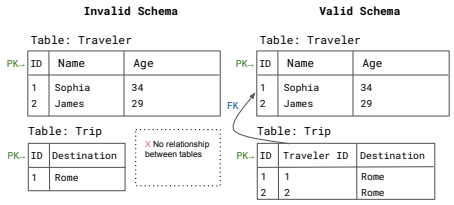


Figure 4: Examples of valid versus invalid relational schemas. PK: Primary key; FK: Foreign key.

**Approach.** The novelty of our approach is to encode a standardized set of rules that reflect the best practices in relational database literature, effectively guiding the model through a structured design process. These rules cover: (1) identifying relevant entities and relationships, (2) defining tables with appropriate columns, (3) assigning primary and foreign keys, and (4) avoiding reserved SQL keywords in naming tables/columns. We encode these rules into two types of prompt strategies: direct, and chain-of-thought (CoT) prompting. CoT decomposes schema generation into intermediate reasoning steps (e.g., entity identification, then table and key definition; see Appendix G).

Decoupling schema generation from tuple formation has another advantage – it allows schema validity to be evaluated in isolation. This modularity is essential for enforcing syntactic constraints: each table must define a primary key (a column, or set of columns that uniquely identifies each row); and tables should include foreign keys (columns referencing primary keys in other tables). These constraints capture relationships between tables and enable `JOIN` operations.

### 3.2 Value Identification

**Challenge.** This stage identifies and extracts values from the text that correspond to columns across all tables in the schema, presenting two challenges. First, multiple values often need to be extracted and deduplicated from the input to form a complete tuple (i.e., an entity instance). In our example, *"Sophia booked a guided tour of Rome with BestCityTours, and opted for the premium package. She was visiting Rome on June 10th."*, we must recover several values, such as traveler name ("Sophia"), tour location ("Rome"), tour operator ("BestCityTours"), and date ("June 10th"); redundant mentions (e.g. "Rome") need to be detected and deduplicated. Second, a document may describe multiple instances of the same type of entity, so we need to assign each value to the correct tuple. For instance, in the passage we also have: *"James, aged 29, was also visiting Rome on June 10th."* Hence, we need to track that Sophia and James are different tourists, and form distinct tuples.

**Approach.** Our neurosymbolic approach first augments direct LLM prompting with two information extraction (IE) methods to isolate values in a structured format, and then guides the LLM to accurately group these values by tuples.

*Triplet Generation.* This step introduces an intermediate representation using *triplets*, a format commonly used in information extraction. Specifically, we consider two triplet formats:

- **Symbolic triplets**, in the form `(subject, relation, object)`—e.g., `(Sophia, visiting, Rome)`, extracted symbolically using the Stanford CoreNLP toolkit (Manning et al., 2014).
- **Schema-aligned triplets**, in the form `(table column, value)`—e.g., `(Tour, Location, Rome)`, generated using prompt-based LLM extraction for the target schema (see Appendix G).

For instance, the earlier passage describing Sophia might yield the following schema-aligned triplets:

| Sophia | James |
|---|---|
| <Traveler, Name, Sophia><br><Trip, Destination, Rome><br><Booking, Date, June 10th> | <Traveler, Name, James><br><Trip, Destination, Rome><br><Booking, Date, June 10th> |

We consider these two types of triplets because each captures complementary sets of values. Symbolic tools use deterministic methods to parse the text, and often extract values that LLMs may overlook (e.g. modifier words like *premium*). In contrast, LLM-generated schema-aligned triplets are more structurally consistent with the database schema, (e.g., *Location* → *Rome*).

To ensure comprehensive coverage, we additionally leverage part-of-speech (POS) tagging to identify all nouns, pronouns, and numerical tokens in the text, since these POS categories typically encompass most values. We then perform string matching to verify whether the extracted triplets include all such tokens. If any are missing, the LLM is prompted to augment the existing triplets by incorporating the missing POS tokens.

*Triplet Deduplication.* Both triplet generation methods often introduce redundancy. To reduce this, we use the "sentence-t5-base" model (Ni et al., 2021) to generate embeddings of the triplets and apply cosine similarity to identify near-duplicates. If a set of triplets has a pairwise cosine similarity above a tunable threshold (97%), we retain only one representative triplet.

*Triplet Grouping.* To ensure that triplets are correctly grouped by entity instance, we apply two heuristics. First, we assume that the first table in the schema typically corresponds to the central entity (e.g., the tourist in a tourism booking system). Second, we leverage the structure of the input document, where each paragraph often describes a distinct instance of this central entity. Accordingly, we associate each paragraph with a unique *identifier*, which serves as the primary key for the first table. In particular, SQUiD uses an LLM to detect the number of distinct entity instances in the document and assign a unique identifier to each paragraph. Once assigned, each triplet is prefixed with its corresponding identifier. For example:

| Sophia | James |
|---|---|
| <1, Traveler, Name, Sophia><br><1, Trip, Destination, Rome><br><1, Booking, Date, June 10th> | <2, Traveler, Name, James><br><2, Trip, Destination, Rome><br><2, Booking, Date, June 10th> |

This structure ensures that all extracted values are correctly grouped by the entity instance they describe, and that the same identifier can be used to link rows across tables during the population stage.

### 3.3 Table Population

**Challenge.** This stage constructs tuples for each table using the values identified in the previous stage, presenting two challenges. First, each value must be correctly *aligned* with its corresponding table column, meaning the LLM must output tuples in a schema-aligned format. However, extracting structured information in a single generation often results in malformed outputs—especially when the target format (e.g., JSON) is complex. Second, we must maintain *referential integrity*: references to the same entity instance must remain consistent *across* related tables. For example, a tuple in the `Trip` table may refer to a destination (e.g., *Rome*) and a traveler (e.g., *Sophia*), who also appears in the `Traveler` table. Here, the traveler ID used in the `Trip` table must match the primary key of the corresponding tuple in the `Traveler` table (Fig. 4).

**Approach.** Before delving into the details, we remind readers that SQUiD has three possible inputs for table population: (1) text alone, (2) text with symbolic triplets, and (3) text with schema-aligned triplets. Including all three in a single prompt increases context length and can degrade output quality. Instead, each source is used independently as input to the prompt, and the resulting tuples are later *combined*. This is akin to *ensemble learning* in ML (Polikar, 2012), allowing us to leverage the complementary strengths of each input.

We now describe the process of table population. To address the value-alignment challenge, we use a structured format that is *incrementally generatable* by the LLM. Instead of emitting the entire structure at once, the format supports iterative generation, which reduces formatting errors. We ensure referential integrity by incorporating carefully chosen guidelines in the prompt that is compatible with the above format. In particular, we leverage *tool*

*use* in LLMs (Qu et al., 2025) by introducing a lightweight tool **extract** that outputs one structured record at a time according to a given schema. This approach helps the LLM remain consistent with the expected output format.

| | |
|---|---|
| Schema | - traveler: "id: int [PK]", "name: string", "age: int"<br>- trip: "id: int [PK]", "traveler_id: int [FK → traveler(id)]", "destination: string", |
| **extract** output | extract traveler: "id": 1; "name": "Sophia"; "age": 34;<br>extract trip: "id": 1; "traveler_id": 1; "destination": "Rome" |

After generating the records, we parse the output to extract each column-value pair for every tuple.

### 3.4 Database Materialization

**Challenge.** A naïve approach is to prompt LLMs with all prior schema and value information to generate the corresponding SQL `INSERT` statements directly. However, this method is both inefficient and error-prone. We observe that this is akin to a "program synthesis" task—it not only requires the production of a large number of redundant tokens, which can be costly; but is also brittle to slight mistakes (e.g., a slightly-malformed SQL statement will produce execution errors).

**Approach.** Instead, we observe that the required SQL statements are well-defined—creating specific tables and then inserting the corresponding tuples to these tables. Therefore, we decouple the materialization step from the LLM by parsing the model's output from the previous stage to *programmatically* construct executable SQL code. Specifically, we generate `CREATE TABLE` and `INSERT INTO` statements (as shown in Fig. 2) which are executed on a local SQLite instance to instantiate the database. This separation enables deterministic parsing, ensuring syntactically correct SQL statements.

## 4 Evaluation Setup

**Dataset.** The Text2R task requires a text document paired with a ground-truth relational database—however, no existing benchmarks directly support this. To fill this gap, we introduce an automated dataset creation pipeline: starting from relational databases or CSV files (using column names and tuple values as ground truth), we prompt an LLM to generate textual descriptions of the tuples, which serve as the input for Text2R. Using this approach, we construct two datasets: (1) **BIRD Dataset**—covering six domains from the BIRD Text2SQL benchmark (Li et al., 2024); and (2) **Kaggle Dataset**—containing CSV files from three domains (*tourism, education, finance*) (Kiattisak, 2023; Becker and Kohavi, 1996; Rai, 2023), which reflect more user-centric, realistic data often missing in BIRD. Table 1 summarizes the dataset statistics. We categorize the text difficulty as **easy** (e.g., *Tourism, Finance*), **medium** (e.g., *Education, California Schools*), or **hard** (e.g., *Mental Health, Superheroes*), based on domain complexity, record sparsity, and LLM-induced verbosity. We also manually annotated samples from the WikiBio (Lebret et al., 2016), CNN-DM (Nallapati et al., 2016), and CORD-19 (Wang et al., 2020) datasets to enable evaluation on real text (see Appendix D.2).

| **Domain** | **Kaggle (24 tables/domain)** | | | **BIRD (24 tables/domain)** | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Tourism | Education | Finance | Calif. Schools | Superhero | Books | Comp. Student | Mental Health | Authors |
| Cols/Table | 12 | 8 | 10 | 26 | 9 | 7 | 6 | 2 | 5 |
| Vals/Table | 60 | 40 | 50 | 130 | 45 | 35 | 30 | 10 | 25 |
| | | | | | | | | **Overall Total Values** | **10,200** |

Table 1: Dataset statistics

**Models.** We test five state-of-the-art models: GPT-4o (OpenAI, 2024), DeepSeek-v2.5 (DeepSeek AI, 2024), Claude 3.7 Sonnet (Anthropic, 2024), LLaMA-3-8B-Instruct (Meta AI, 2024), and Qwen3-8B (Alibaba, 2024).

**Metrics.** We propose a suite of novel metrics for a principled evaluation of the Text2R task, which are summarized in Table 2.

*Schema Evaluation.* We evaluate the quality of generated database schemas along three dimensions: **entity coverage**, **primary key coverage**, and **foreign key coverage**. Entity coverage assesses whether each column from the ground truth is represented in the generated schema. A column is considered covered if there exists a semantically equivalent column (based on cosine similarity between column names) in the output. Primary key coverage checks whether each generated table defines at least one primary key, while foreign key coverage evaluates whether all foreign keys correctly reference primary keys in valid, related tables within the schema. The last two metrics assess syntactic constraints that are essential for the correctness of relational database schemas.

*Tuple Evaluation.* Relational databases store data across multiple tables; therefore, evaluating the quality of such databases requires a holistic view that goes beyond individual tables or isolated values. To enable a principled evaluation, we flatten the schema into a single table—commonly referred to as a denormalized table (Elmasri and Navathe, 2016)—by performing a `JOIN` across all tables. In our databases, each table maintains a many-to-one or one-to-one relationship with a central table, enabling this complete `JOIN` of the entire schema. This consolidated table captures complete entity-relationship instances in a unified format. We

| | Evaluation Metrics | Definition | Formula |
|---|---|---|---|
| Schema | Entity Coverage Score (ECS) | Avg. max cosine similarity betn. GT & DB columns | $\frac{1}{N}\sum_{i=1}^{N} \max_{j\in[1,M]} \texttt{cos\_sim}(c_i, \hat{c}_j)$ |
| | Primary Key Coverage (PKC) | % of tables with a defined primary key | $\#\texttt{Tables with PK}/\#\texttt{Tables}$ |
| | Foreign Key Coverage (FKC) | % of tables whose foreign keys refer valid primary keys | $\#\texttt{Tables with valid FK}/\#\texttt{Tables}$ |
| Tuple | Database Construction Success Rate (DBR) | % of successfully generated databases | $\#\texttt{Generated DB}/\#\texttt{Text Documents}$ |
| | Tuple Coverage (TC) | % of GT tuples present in DB | $\lvert\mathcal{R}_{\texttt{GT}}\cap\mathcal{R}_{\texttt{DB}}\rvert/\lvert\mathcal{R}_{\texttt{GT}}\rvert$ |
| | Value Coverage (VC) | % of GT values present in DB | $\lvert\mathcal{V}_{\texttt{GT}}\cap\mathcal{V}_{\texttt{DB}}\rvert/\lvert\mathcal{V}_{\texttt{GT}}\rvert$ |
| | Column Consistency (CC) | % of GT values present in DB in correct columns | $\lvert\mathcal{V}_{\texttt{GT}}\cap\mathcal{V}_{\texttt{DB}}\rvert/\lvert\mathcal{V}_{\texttt{GT}}\rvert$, where $\mathcal{V}_{\texttt{GT}}, \mathcal{V}_{\texttt{DB}} \in \texttt{col}$ |
| | Ref. Integrity % (RRIR) | Avg. tuple completeness after FK joins (non-null ratio) | $\texttt{RRIR} = \frac{1}{N}\sum_{i=1}^{N} \frac{\#\texttt{non-null values in tuple } i}{\#\texttt{total values in tuple } i}$ |

Table 2: Novel evaluation metrics for Text2R: GT denotes ground truth and DB denotes the generated databases.

generate two denormalized tables: one from the ground-truth database and one from the database produced by SQUiD. The two are then compared to assess the accuracy of the generated database.

We propose five novel metrics to evaluate the quality of the generated tuples along two dimensions: syntactic and semantic validity. Syntactic validity assesses whether the generated databases adhere to correct structural and relational rules. It is measured using: (1) **Database Construction Success Rate**, which measures the percentage of generated SQL statements that successfully materialize into databases with at least one non-null tuple, (2) **Referential Integrity Rate** (RRIR), which measures the fraction of foreign-key joins that yielded valid (non-null) tuples.

Semantic validity evaluates the comprehensiveness and correctness of the values populated. It is measured using: (1) **Tuple Coverage**, which measures the fraction of the ground truth tuples recovered; (2) **Value Coverage**, which measures the fraction of ground truth values populated; and (3) **Column Consistency**, which checks whether each value appears in its correct column.

**Baseline.** We design a tailored baseline: using zero-shot prompting, we generate `CREATE TABLE` and `INSERT INTO` SQL statements directly from the input text, then execute them in SQLite to instantiate the database, serving as the most relevant point of reference. Prompt details are in Appendix G.

Existing work, by contrast, addresses a fundamentally different task—text-to-table summarization—which diverges from our task in the following key aspects: (1) Text2R involves generating interrelated tables which must satisfy constraints such as primary/foreign keys, whereas prior work produces independent tables that cannot be queried as a database. (2) Text2R requires consistent value population across related tables—e.g., the same entity must be consistently represented wherever it occurs. Existing work lacks this requirement as their tables are unrelated. (3) Database creation requires valid, executable SQL statements. This critical step is entirely absent in prior work, which generates markdown-style tables (i.e., plain text), not databases. Instead, SQUiD generates fully executable SQLite databases in the user's system, providing a way to perform computation over unstructured text, rather than mere text summarization. Thus, any end-to-end quantitative comparison with prior work on Text2R would not be meaningful (as highlighted in Fig. 3). Nevertheless, for completeness, we adapt prompts from T3 (Deng et al., 2024), STRUCTSUM (Jain et al., 2024), and EVAPORATE (Arora et al., 2023), and evaluate them on a subset of our metrics where a fair comparison is possible. Since these methods output only markdown tables, we relied on custom parsing to extract column names and values, which introduces some unreliability. In contrast, SQUiD supports principled evaluation through relational databases and query-based retrieval.

## 5 Experiments and Analysis

We evaluate the performance of SQUiD based on the following three research questions (RQs):

- **RQ1.** Can SQUiD generate a high-quality relational schema?
- **RQ2.** Can SQUiD generate accurate relational tuples to populate the tables?
- **RQ3.** How do SQUiD's design choices affect performance?
- **RQ4.** How does SQUiD's performance compare to prior work?

### 5.1 RQ1: Schema Evaluation

As described in Sec.3.1, we evaluate two prompting strategies for schema generation: Direct and Chain-of-Thought (CoT). Table 3 summarizes the results. We only consider schemas that match the format specified in the prompt, as this is required for SQUiD to process them later. We evaluate both syntactic validity—using primary key coverage (PKC) and foreign key coverage (FKC)—and semantic validity, using entity coverage (ECS). We first highlight general observations across all three metrics, followed by specific analysis. Overall,

| Model | Prompt | Easy | | | Medium | | | Hard | | | Avg. | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ECS(%) | PKC(%) | FKC(%) | ECS(%) | PKC(%) | FKC(%) | ECS(%) | PKC(%) | FKC(%) | ECS(%) | PKC(%) | FKC(%) |
| CLAUDE 3.7 SONNET | Direct | 86.2 | 100 | 100 | 80.7 | 100 | 100 | 45.3 | 100 | 100 | 70.7 | 100 | 100 |
| LLAMA-8B INSTRUCT | Direct | 80.4 | 100 | 100 | 78.6 | 100 | 100 | 55.4 | 100 | 100 | 71.5 | 100 | 100 |
| | CoT | 95.8 | 100 | 100 | 76.5 | 100 | 100 | 62.7 | 100 | 100 | 78.4 | 100 | 100 |
| DEEPSEEK V2.5 | Direct | – | – | – | 28.3 | 33.33 | 33.3 | 34.5 | 50 | 50 | 20.9 | 27.8 | 27.8 |
| | CoT | 86.9 | 100 | 100 | 84.2 | 100 | 100 | 65.1 | 100 | 100 | 78.8 | 100 | 100 |
| GPT-4O | Direct | 90.5 | 100 | 100 | 79.4 | 94.4 | 100 | 62.6 | 100 | 100 | 77.5 | 98.2 | 100 |
| | CoT | 93.0 | 100 | 100 | 80.0 | 100 | 66.7 | 63.2 | 100 | 100 | 78.7 | 100 | 88.9 |

Table 3: Schema evaluation: Entity (ECS), Primary Key (PKC) and Foreign Key (FKC) coverage scores. "–": schema generation failures that violate the requested structure in our prompts. CLAUDE-CoT and QWEN-8B are omitted due to such failures.

CoT consistently outperforms Direct across difficulty levels; except CLAUDE, which performs better with Direct but struggles with CoT due to format violations, likely due to overthinking (Liu et al., 2024b). QWEN-8B consistently fails to produce valid schemas, likely due to poor support for structured output tasks (Liu et al., 2024c).

**Syntactic Validity.** We observe that most CoT-based generations achieve full PKC and FKC, except GPT, which drops to 66.67% FKC in the medium dataset. This is because GPT occasionally generates a single table with no foreign key, when the text contains only a few entities.

**Semantic Validity.** For entity coverage ECS, DEEPSEEK with CoT performs the best, followed by LLAMA-8B and GPT–which show minor drops due to their tendency to generate paraphrased column names (e.g., "heritage" or "ethnicity" instead of "race"), whereas DEEPSEEK aligns more closely with the ground truth. In terms of performance across domains (Appendix D), DEEPSEEK achieves the highest entity coverage in the *Education* domain (91.08%) and the lowest in the *Mental Health* domain (38.97%). The ground truth of the latter has complex column names, such as "questiontext" and "answertext", suggesting that domain complexity significantly affects the quality of the generated schema.

## 5.2 RQ2: Tuple Evaluation

**Syntactic Validity.** Table 4 reports the Database Construction Success Rate (DBR) and the improvement in Referential Integrity Rate (RRIR) over the baseline. We highlight three observations. First, SQUiD achieves perfect DBR (100%) across all models and difficulty levels, except for using DEEPSEEK on hard examples, where it drops slightly to 98%. This indicates the robustness of SQUiD in consistently generating syntactically valid databases. We observe similar robustness of SQUiD in real text datasets (see Appendix. D.2). In contrast, the baseline DBR varies widely—from as low as 9.7% (GPT) to 58.2% (CLAUDE) on average. Next, we turn to referential integrity. We note that SQUiD 's RRIR is a conservative (lower-bound) estimate, since records with missing values in the ground truth are treated as invalid under our metric. Nevertheless, SQUiD still achieves significant improvements over the baseline. For example, GPT exhibits the highest improvement ($46.59\times$ on easy examples). QWEN-8B also achieve notable average improvements of $3.52\times$. Although LLAMA-8B achieves perfect DBR, its RRIR does not improve on the medium dataset, suggesting its baseline already exhibits relatively strong referential integrity.

| | | DBR(%) | | RRIR |
|---|---|---|---|---|
| | | SQUiD | Baseline | Improvement Factor |
| CLAUDE 3.7 SONNET | Easy | 100.0 | 63.2 | 1.56× |
| | Medium | 100.0 | 63.4 | 1.10× |
| | Hard | 100.0 | 48.1 | 1.41× |
| | Average | 100.0 | 58.2 | 1.40× |
| DEEPSEEK V2.5 | Easy | 100.0 | 23.2 | 4.44× |
| | Medium | 100.0 | 42.4 | 1.70× |
| | Hard | 98.0 | 40.3 | 1.87× |
| | Average | 99.3 | 35.3 | 1.80× |
| GPT-4O | Easy | 100.0 | 2.0 | 46.59× |
| | Medium | 100.0 | 6.1 | 12.09× |
| | Hard | 100.0 | 21.0 | 2.63× |
| | Average | 100.0 | 9.7 | 13.93× |
| QWEN3 -8B | Easy | 100.0 | 23.5 | 4.42× |
| | Medium | 100.0 | 32.2 | 2.52× |
| | Hard | 100.0 | 10.4 | 6.83× |
| | Average | 100.0 | 22.0 | 3.52× |
| LLAMA-3 8B-INSTRUCT | Easy | 100.0 | 63.2 | 1.54× |
| | Medium | 100.0 | 64.5 | 1.00× |
| | Hard | 100.0 | 40.1 | 1.87× |
| | Average | 100.0 | 55.9 | 1.64× |

Table 4: Database Construction Success Rate (%) and the improvement factor in Referential Integrity Rate in SQUiD compared to the baseline.

**Semantic Validity.** Table 5 reports Tuple Coverage (TC), Value Coverage (VC), and Column Consistency (CC) with three findings. First, SQUiD consistently outperforms the baseline across all models and metrics. Notably, all 8B-parameter models (LLAMA-8B, QWEN-8B) under SQUiD signifi-

cantly outperform all larger model baselines (GPT, CLAUDE, DEEPSEEK). In particular, although QWEN-8B's baseline lags behind those of CLAUDE and DEEPSEEK, its performance under SQUiD surpasses them—highlighting the effectiveness of our approach. Second, on average, all models using SQUiD achieve high TC ($\geq$0.95) and strong VC/CC ($\geq$0.70), with GPT showing the largest improvement over its baseline (17.75× improvement on CC). This is primarily because failed database generations are assigned zero scores, and as shown in Table 4, GPT performs poorly in database construction under the baseline setting. Third, even for models with relatively strong baseline performance, such as LLAMA-8B, SQUiD improves VC and CC by 4.1× and 5.5× on hard examples, respectively.

| Model | Diff. | SQUiD | | | Baseline | | |
|---|---|---|---|---|---|---|---|
| | | TC(%) | VC(%) | CC(%) | TC(%) | VC(%) | CC(%) |
| CLAUDE 3.7 | Easy | 100.0 (2.56×) | 98.0 (4.67×) | 98.0 (4.67×) | 39.0 | 21.0 | 21.0 |
| | Med | 98.0 (2.72×) | 78.0 (6.00×) | 74.0 (5.69×) | 36.0 | 13.0 | 13.0 |
| | Hard | 100.0 (2.44×) | 63.0 (2.74×) | 41.0 (3.73×) | 41.0 | 23.0 | 11.0 |
| | Avg | 99.0 (2.61×) | 80.0 (4.21×) | 71.0 (4.73×) | 38.0 | 19.0 | 15.0 |
| DEEPSEEK-V2.5 | Easy | 100.0 (5.88×) | 96.0 (6.86×) | 96.0 (6.86×) | 17.0 | 14.0 | 14.0 |
| | Med | 99.0 (3.54×) | 80.0 (5.33×) | 77.0 (5.50×) | 28.0 | 15.0 | 14.0 |
| | Hard | 95.0 (2.64×) | 59.0 (2.57×) | 39.0 (3.90×) | 36.0 | 23.0 | 10.0 |
| | Avg | 98.0 (3.63×) | 79.0 (4.65×) | 71.0 (5.92×) | 27.0 | 17.0 | 12.0 |
| GPT-4O | Easy | 100.0 (50.00×) | 97.0 (48.50×) | 97.0 (48.50×) | 2.0 | 2.0 | 2.0 |
| | Med | 99.0 (16.50×) | 81.0 (16.20×) | 77.0 (19.25×) | 6.0 | 5.0 | 4.0 |
| | Hard | 97.0 (6.47×) | 61.0 (5.55×) | 40.0 (6.67×) | 15.0 | 11.0 | 6.0 |
| | Avg | 99.0 (**14.14**×) | 80.0 (**13.33**×) | 71.0 (**17.75**×) | 7.0 | 6.0 | 4.0 |
| LLAMA3-8B-IN. | Easy | 100.0 (1.82×) | 95.0 (3.06×) | 95.0 (3.17×) | 55.0 | 31.0 | 30.0 |
| | Med | 99.0 (1.83×) | 79.0 (2.82×) | 75.0 (3.00×) | 54.0 | 28.0 | 25.0 |
| | Hard | 100.0 (3.45×) | 70.0 (4.12×) | 44.0 (5.50×) | 29.0 | 17.0 | 8.0 |
| | Avg | **100.0** (2.17×) | **81.0** (3.24×) | 71.0 (3.38×) | 46.0 | 25.0 | 21.0 |
| QWEN3-8B | Easy | 100.0 (4.55×) | 96.0 (5.05×) | 96.0 (5.33×) | 22.0 | 19.0 | 18.0 |
| | Med | 98.0 (3.27×) | 79.0 (3.16×) | 79.0 (3.43×) | 30.0 | 25.0 | 23.0 |
| | Hard | 99.0 (14.14×) | 51.0 (10.20×) | 51.0 (17.00×) | 7.0 | 5.0 | 3.0 |
| | Avg | 99.0 (4.95×) | 76.0 (4.75×) | **75.0** (5.00×) | 20.0 | 16.0 | 15.0 |

Table 5: Tuple evaluation via Tuple Coverage (TC), Value Coverage (VC) and Column Consistency (CC). Best scores and improvement factors across models in **bold**. Gray indicates that SQUiD on all 8B models outperforms larger models.

### 5.3 RQ3: Impact of SQUiD's Design Choices

We now evaluate the impact of SQUiD's design choices on value identification and table population. Recall that we consider three different prompts for table population based on their input source: (1) text only ($\mathbb{T}$), (2) text with symbolic triplets ($\mathbb{S}$), and (3) text with schema-aligned triplets ($\mathbb{L}$). SQUiD combines the rows generated from all three prompts. Table 6 evaluates how these different value sources affect the quality of the generated tuples, with the following observations.

First, using triplets significantly improves value coverage compared to extracting them from the text alone. This is evident from the observation

| Model | Diff. | $\mathbb{T}$ (%) (1) | $\mathbb{S}$ (%) (2) | $\mathbb{L}$ (%) (3) | $\mathbb{T}\oplus\mathbb{S}$ (%) (1)+(2) | $\mathbb{T}\oplus\mathbb{L}$ (%) (1)+(3) | SQUiD (%) (1)+(2)+(3) |
|---|---|---|---|---|---|---|---|
| CLAUDE 3.7 | Easy | 97.4 | 97.1 | 93.8 | 98.3 | 97.7 | 98.4 |
| | Med | 68.2 | 74.6 | 74.1 | 77.3 | 77.5 | 78.2 |
| | Hard | 51.7 | 58.4 | 51.3 | 60.7 | 60.5 | 63.1 |
| | Avg | 72.4 | 76.7 | 73.1 | 78.8 | 78.6 | 79.9 |
| DEEPSEEK-V2.5 | Easy | 92.3 | 94.5 | 92.7 | 96.8 | 95.4 | 96.8 |
| | Med | 76.7 | 68.1 | 69.3 | 79.6 | 80.2 | 80.4 |
| | Hard | 54.8 | 42.9 | 35.7 | 57.4 | 57.1 | 59.3 |
| | Avg | 74.6 | 68.5 | 65.9 | 77.9 | 77.6 | 78.8 |
| GPT-4O | Easy | 90.8 | 93.2 | 90.4 | 95.1 | 96.3 | 97.4 |
| | Med | 75.3 | 69.7 | 68.6 | 80.4 | 81.2 | 81.3 |
| | Hard | 50.6 | 41.8 | 51.7 | 56.3 | 59.4 | 61.6 |
| | Avg | 72.2 | 68.2 | 70.2 | 77.3 | 79.0 | 80.1 |
| LLAMA3-8B-IN. | Easy | 89.3 | 74.8 | 61.5 | 95.2 | 94.4 | 95.5 |
| | Med | 70.1 | 52.7 | 61.3 | 75.5 | 76.1 | 79.4 |
| | Hard | 60.6 | 37.9 | 40.4 | 64.3 | 68.5 | 70.7 |
| | Avg | 73.3 | 55.1 | 54.4 | 78.3 | 79.7 | 81.9 |
| QWEN3-8B | Easy | 92.1 | 92.6 | 72.5 | 96.4 | 96.1 | 96.4 |
| | Med | 71.4 | 71.3 | 67.4 | 74.7 | 78.4 | 79.5 |
| | Hard | 29.8 | 23.5 | 35.9 | 33.3 | 48.2 | 51.2 |
| | Avg | 64.4 | 62.5 | 58.6 | 68.1 | 74.2 | 75.7 |

Table 6: Impact of different value source. The first three columns represent individual prompt settings, while the last three correspond to post-generation ensembling. $\mathbb{T}\oplus\mathbb{S}$ combines tuples generated from $\mathbb{T}$ and $\mathbb{S}$ while $\mathbb{T}\oplus\mathbb{L}$ combines $\mathbb{T}$ and $\mathbb{L}$. SQUiD combines outputs from all three prompts.

that SQUiD outperforms $\mathbb{T}$ by 5–12%.

Second, we examine how to best incorporate the triplets: whether to concatenate them with the input text in a *single* prompt, or to generate tuples separately and combine them post-hoc (ensembling). SQUiD adopts the latter strategy, and our results support this choice. Specifically, in the individual prompt setting, $\mathbb{T}$ outperforms both $\mathbb{S}$ and $\mathbb{L}$ in all but one case (CLAUDE). In contrast, the ensemble approaches ($\mathbb{T}\oplus\mathbb{S}$ , $\mathbb{T}\oplus\mathbb{L}$ and SQUiD) consistently outperform all the individual prompts. This suggests that including triplets directly in the input prompt increases context length, which degrades model performance—likely due to context window saturation (Liu et al., 2024a).

Finally, we evaluate our design choice of combining triples generated from symbolic tools and schema-aligned triplets from LLMs. Overall, $\mathbb{T}\oplus\mathbb{L}$ outperforms $\mathbb{T}\oplus\mathbb{S}$ across most models on average, except for CLAUDE and DEEPSEEK. SQUiD consistently yields the best score, indicating that each source captures *complementary* information. LLM-generated triplets are schema-aware and can correctly group multi-word values under the correct columns (e.g., mapping "car rental" to the *transportation mode* column, whereas symbolic tools only captured "car"). However, LLMs sometimes paraphrase values (e.g., "low income" to "modest

income"), whereas symbolic tools extract values verbatim, yielding closer alignment to the input.

### 5.4 RQ4: Comparison of SQUiD's Performance with Related Work

| Model | Diff. | T3 | | Evaporate | | StructSum | | SQUiD | |
|---|---|---|---|---|---|---|---|---|---|
| | | VC (%) | CC (%) | VC (%) | CC (%) | VC (%) | CC (%) | VC (%) | CC (%) |
| CLAUDE 3.7 | Easy | 12.7 | 2.3 | 55.1 | 42.8 | 61.0 | 52.0 | 98.0 (×**7.72**/1.61/1.60) | 98.0 (×**42.6**/2.29/1.88) |
| | Med | 19.3 | 1.8 | 46.4 | 34.1 | 50.0 | 41.5 | 78.0 (×**4.04**/1.68/1.56) | 74.0 (×**41.1**/2.17/1.78) |
| | Hard | 25.5 | 0.9 | 15.6 | 7.2 | 62.5 | 23.5 | 63.0 (×2.47/**4.04**/1.01) | 41.0 (×**45.6**/5.69/1.74) |
| | Avg | 19.2 | 1.7 | 39.0 | 28.0 | 57.8 | 39.0 | 80.0 (×**4.17**/2.05/1.38) | 71.0 (×**41.8**/2.54/1.82) |
| DEEPSEEK-V2.5 | Easy | 8.9 | 1.6 | 60.2 | 54.8 | 61.9 | 55.2 | 96.0 (×**10.8**/1.59/1.55) | 96.0 (×**60.0**/1.75/1.74) |
| | Med | 13.8 | 4.3 | 50.0 | 40.4 | 50.0 | 44.2 | 80.0 (×**5.80**/1.60/1.60) | 77.0 (×**17.9**/1.91/1.74) |
| | Hard | 4.8 | 1.3 | 10.7 | 6.9 | 63.1 | 27.4 | 59.0 (×**12.3**/5.51/0.94) | 39.0 (×**30.0**/5.65/1.42) |
| | Avg | 9.2 | 2.4 | 40.3 | 34.0 | 58.3 | 42.3 | 79.0 (×**8.59**/1.96/1.35) | 71.0 (×**29.6**/2.09/1.68) |
| LLAMA3-8B-IN. | Easy | 13.0 | 2.1 | 59.7 | 40.1 | 60.0 | 55.2 | 95.0 (×**7.31**/1.59/1.58) | 95.0 (×**45.2**/2.37/1.72) |
| | Med | 13.9 | 1.2 | 48.9 | 35.5 | 50.0 | 42.3 | 79.0 (×**5.68**/1.61/1.58) | 75.0 (×**62.5**/2.11/1.77) |
| | Hard | 11.8 | 0.9 | 53.6 | 3.5 | 61.9 | 14.3 | 70.0 (×**5.93**/1.31/1.13) | 44.0 (×**48.9**/12.6/3.08) |
| | Avg | 12.9 | 1.4 | 54.1 | 26.4 | 57.3 | 37.3 | 81.0 (×**6.28**/1.49/1.41) | 71.0 (×**50.7**/2.69/1.90) |
| GPT-4O | Easy | 14.5 | 3.9 | 33.6 | 25.5 | 60.5 | 51.2 | 97.0 (×**6.69**/2.89/1.60) | 97.0 (×**24.9**/3.80/1.89) |
| | Med | 26.2 | 2.4 | 41.5 | 25.6 | 50.0 | 40.4 | 81.0 (×**3.09**/1.95/1.62) | 77.0 (×**32.1**/3.01/1.91) |
| | Hard | 43.6 | 3.5 | 2.4 | 1.2 | 64.3 | 28.6 | 61.0 (×1.40/**25.4**/0.95) | 40.0 (×11.4/**33.3**/1.40) |
| | Avg | 28.1 | 3.3 | 25.8 | 17.4 | 58.3 | 40.1 | 80.0 (×**2.85**/3.10/1.37) | 71.0 (×**21.5**/4.08/1.77) |
| QWEN3-8B | Easy | 11.4 | 2.2 | 58.7 | 43.9 | 61.2 | 55.0 | 96.0 (×**8.42**/1.63/1.56) | 96.0 (×**43.6**/2.19/1.75) |
| | Med | 15.9 | 1.1 | 47.3 | 33.0 | 50.0 | 43.0 | 79.0 (×**4.97**/1.67/1.58) | 79.0 (×**71.8**/2.39/1.84) |
| | Hard | 9.2 | 0.6 | 21.5 | 12.0 | 62.0 | 21.5 | 51.0 (×**5.54**/2.37/0.82) | 51.0 (×**85.0**/4.25/2.37) |
| | Avg | 12.2 | 1.3 | 42.5 | 29.6 | 57.7 | 39.8 | 75.0 (×**6.15**/1.76/1.30) | 75.0 (×**57.7**/2.53/1.88) |

Table 7: Inline multipliers indicate SQUiD 's improvement factors over T3, Evaporate, and StructSum (in that order). The best improvement factor is highlighted in **bold**.

We design a new baseline for SQUiD, since prior work cannot serve as a direct point of comparison—the tasks differ fundamentally and do not allow for an apples-to-apples evaluation. Nevertheless, for completeness, we report comparison results in Table 7 on a subset of metrics that remain meaningful. Schema-level metrics and DBR are excluded, as they assess the syntactic validity of relational databases, whereas prior methods output non-relational tables. TC is also not a fair metric: for a single flat table, it's straightforward to enumerate all the tuples. In contrast, SQUiD generates multiple interrelated tables, introducing two key challenges: ensuring consistent representation of entities across tables and retrieving attributes spread among them. Reconstructing the correct tuples for the normalized evaluation database (via JOIN; see Sec. 4) is therefore non-trivial. The only fair metrics for comparison are VC and CC, which test whether all ground-truth values appear in the output and whether they are placed in the correct columns, respectively. As shown in Table 7, SQUiD substantially outperforms all baselines, achieving the best scores across all models and difficulty levels (up to an **85×** improvement on the Hard dataset for CC, Qwen3-8B). CC is especially poor for T3 prompts, with SQUiD yielding on average a **42×** improvement over T3 across models. We attribute SQUiD's substantial performance gains to a fundamental limitation of prior methods: markdown tables lack atomicity. Designed for summarization, they often compress multiple attributes into a single column (e.g., merging *hotel name* and *cost* under *accommodation*), resulting in misalignment with the ground truth. By contrast, SQUiD preserves the atomicity of relational databases by first defining an explicit schema, ensuring that each value occupies its own column and that all relevant information is captured in full rather than summarized.

## 6 Related Work

**Summarizing Structures.** *Text-to-table* generation (Wu et al., 2022; Sundar et al., 2024; Li et al., 2023; Deng et al., 2024; Arora et al., 2023; Jain et al., 2024) projects explore sequence-to-sequence modeling, LLM prompt engineering, and structured summarization techniques. However, they can only generate flat tables, and cannot capture the relational database model in our work.

**Manipulating Existing Databases.** The goal of these projects is to leverage LLMs to interact with existing relational databases—such as to generate SQL queries from text (Hong et al., 2024; Pang et al., 2020), or to update them using natural language (Jiao et al., 2024). However, none of these works can synthesize a relational database from scratch, which is what SQUiD tackles.

**Non-LLM Approaches.** Prior to LLMs, integrating text into relational structures relied on traditional pipelines that combine information extraction, schema induction, and entity linking (Zhang et al., 2016; Smith et al., 2022b; Zhang et al., 2019). These methods rely on statistical or symbolic techniques, but required domain-specific heuristics and did not generalize to noisy or diverse input text.

## 7 Conclusion

In this work, we have introduced a novel task of synthesizing relational databases from text, called Text2R. We have also developed a framework, SQUiD, designed to solve Text2R tasks. SQUiD has a neurosymbolic pipeline, with each stage incorporating specialized techniques for the task. Our experiments show SQUiD significantly outperforms baseline solutions across diverse datasets.

### Limitations

While we provide extensive evaluation of SQUiD on our benchmark, we leave comparisons with few-shot baselines and fine-tuned models for future work.

## Ethics Statement

All datasets used in this work are publicly available and released under open licenses. The tools and models employed are authorized for research purposes and have been used in accordance with their intended terms. Detailed license information is provided in Appendix F. All experiments were performed strictly for research and evaluation.

Because our study requires user-centric documents for schema generation and value mapping evaluation, anonymization was not feasible without significantly compromising data integrity. To the best of the authors' knowledge, this research does not introduce any ethical risks beyond those already associated with the original datasets.

Since SQUiD uses large language models (LLMs) to synthesize databases, and LLMs are known to occasionally produce hallucinated or inaccurate content, there are potential risks when applying SQUiD in sensitive domains without human oversight. Careful review and verification are recommended before deploying the system in high-stakes or privacy-critical applications.


## Acknowledgments

This work is partially supported by NSF grants CNS-2535540, CNS-2406598, CNS-2420309, and CNS-2345339.


## References


Alibaba. 2024. Qwen3 language models. Accessed: 2025-05-19. Licensed under the Apache 2.0 License.

Anthropic. 2024. Claude 3 model family. Accessed: 2025-05-19. Usage governed by Anthropic's terms of service.

Simran Arora, Brandon Yang, Sabri Eyuboglu, Avanika Narayan, Andrew Hojel, Immanuel Trummer, and Christopher Ré. 2023. Language models enable simple systems for generating structured views of heterogeneous data lakes. *Proc. VLDB Endow.*, 17(2):92–105.

Wen Bai, Shuo Liu, and Kai Zhang. 2023. Schema-driven information extraction from heterogeneous tables. *arXiv preprint arXiv:2306.12345*.

Barry Becker and Ronny Kohavi. 1996. Adult. UCI Machine Learning Repository. DOI: https://doi.org/10.24432/C5XW20.

Andrew Carlson and Charles Schafer. 2008. Bootstrapping information extraction from semi-structured web pages. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML PKDD)*.

Chia-Hui Chang and Chun-Ying Wu. 2016. Fastwrapper: Learning structure from web pages for data extraction. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining (PAKDD)*.

Jiayi Chang, Yu Liu, and Fang Chen. 2024. Synthesizing text-to-sql data from weak and strong llms. In *Proceedings of the 2024 Annual Meeting of the Association for Computational Linguistics (ACL)*.

Eric Chu, Akanksha Baid, Ting Chen, AnHai Doan, and Jeffrey Naughton. 2007. A relational approach to incrementally extracting and querying structure in unstructured data. In *Proceedings of the 33rd International Conference on Very Large Data Bases (VLDB)*, pages 1045–1056.

DeepSeek AI. 2024. Deepseek-v2.5: A next-generation language model. Accessed: 2025-05-19. Licensed under the DeepSeek License.

Xi Deng. 2010. Automatic web data extraction using tree matching and partial tree alignment. In *Proceedings of IEEE International Conference on Data Engineering Workshops (ICDE Workshops)*.

Xi Deng. 2011. Sede: Schema extraction from html data sources. In *IEEE International Conference on Data Engineering (ICDE)*.

Zheye Deng, Chunkit Chan, Weiqi Wang, Yuxi Sun, Wei Fan, Tianshi Zheng, Yauwai Yim, and Yangqiu Song. 2024. Text-tuple-table: Towards information integration in text-to-table generation via global tuple extraction.

Ramez Elmasri and Shamkant B. Navathe. 2016. *Fundamentals of Database Systems*, 7 edition. Pearson.

Tam Harbert. n.d. Tapping the power of unstructured data. `https://mitsloan.mit.edu/ideas-made-to-matter/tapping-power-unstructured-data`.

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 preprint arXiv:2406.08426*.

Alpa Jain, AnHai Doan, and Luis Gravano. 2007. Sql queries over unstructured text databases. In *2007 IEEE 23rd International Conference on Data Engineering (ICDE)*, pages 1255–1257.

Parag Jain, Andreea Marzoca, and Francesco Piccinno. 2024. STRUCTSUM generation for faster text comprehension. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7876–7896, Bangkok, Thailand. Association for Computational Linguistics.

Yizhu Jiao, Sha Li, Sizhe Zhou, Heng Ji, and Jiawei Han. 2024. Text2DB: Integration-aware information extraction with large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 185–205, Bangkok, Thailand. Association for Computational Linguistics.

Ratanakorn Kiattisak. 2023. Traveler trip data. Accessed: 2025-05-07.

Rémi Lebret, David Grangier, and Michael Auli. 2016. Neural text generation from structured data with application to the biography domain. *arXiv preprint arXiv:1603.07771*.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Mingda Li, Yichong Chen, and Jiawei Han. 2023. Seq2seqset: Modular table generation via sequential header and set-based body construction. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.

Nelson F. Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024a. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.

Ryan Liu, Jiayi Geng, Addison J. Wu, Ilia Sucholutsky, Tania Lombrozo, and Thomas L. Griffiths. 2024b. Mind your step (by step): Chain-of-thought can reduce performance on tasks where thinking makes humans worse.

Yu Liu, Duantengchuan Li, Kaili Wang, Zhuoran Xiong, Fobo Shi, Jian Wang, Bing Li, and Bo Hang. 2024c. Are llms good at structured outputs? a benchmark for evaluating structured output capabilities in llms. *Information Processing and Management*, 61(5):103809.

Christopher Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven Bethard, and David McClosky. 2014. The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 55–60, Baltimore, Maryland. Association for Computational Linguistics.

I.R. Mansuri and S. Sarawagi. 2006. Integrating unstructured data into relational databases. In *22nd International Conference on Data Engineering (ICDE'06)*, pages 29–29.

Meta AI. 2024. Llama 3: Open foundation and instruction-tuned language models. Accessed: 2025-05-19. Licensed under Meta's LLaMA 3 Community License.

Matthew Michelson and Craig A. Knoblock. 2008. Creating relational data from unstructured and ungrammatical data sources. *Journal of Artificial Intelligence Research*, 31:543–590.

Karin Murthy, Prasad M. Deshpande, Atreyee Dey, Ramanujam Halasipuram, Mukesh Mohania, P. Deepak, Jennifer Reed, and Scott Schumacher. 2012. Exploiting evidence from unstructured data to enhance master data management. *Proceedings of the VLDB Endowment*, 5(12):1862–1873.

Ramesh Nallapati, Bowen Zhou, Caglar Gulcehre, Bing Xiang, et al. 2016. Abstractive text summarization using sequence-to-sequence rnns and beyond. *arXiv preprint arXiv:1602.06023*.

Jianmo Ni, Gustavo Hernández Ábrego, Noah Constant, Ji Ma, Keith B. Hall, Daniel Cer, and Yinfei Yang. 2021. Sentence-t5: Scalable sentence encoders from pre-trained text-to-text models.

Christina Niklaus, Matthias Cetto, André Freitas, and Siegfried Handschuh. 2018. A survey on open information extraction. *arXiv preprint arXiv:1806.05599*.

Madhav Nimishakavi and Partha Talukdar. 2016. Relation schema induction using tensor factorization with side information. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

OpenAI. 2024. Gpt-4o technical report. Accessed: 2025-05-19. Usage governed by OpenAI's terms of service.

Long Pang, Tao Zhang, and Ming Hu. 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 (ACL)*.

Robi Polikar. 2012. Ensemble learning. *Ensemble Machine Learning: Methods and Applications*, pages 1–34.

Changle Qu, Sunhao Dai, Xiaochi Wei, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, Jun Xu, and Ji-rong Wen. 2025. Tool learning with large language models: A survey. *Frontiers of Computer Science*, 19(8).

Harun Rai. 2023. Fintech customer life time value (ltv) dataset. Accessed: 2025-05-07.

Thomas Scholak, Siva Reddy Patra, and James Compositionality. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Hassan Shavarani and Anoop Sarkar. 2025. Entity retrieval for answering entity-centric questions. In *Proceedings of the 4th International Workshop on Knowledge-Augmented Methods for Natural Language Processing*, pages 1–17, Albuquerque, New Mexico, USA. Association for Computational Linguistics.

Abraham Silberschatz, Henry F. Korth, and S. Sudarshan. 2020. *Database System Concepts*, 7 edition. McGraw-Hill Education.

Ellery Smith, Dimitris Papadopoulos, Martin Braschler, and Kurt Stockinger. 2022a. Lillie: Information extraction and database integration using linguistics and learning-based algorithms. *Information Systems*, 105:101938.

Jack Smith, Evangelos Kanoulas, et al. 2022b. Lillie: Language-independent linked information extraction. *Data and Knowledge Engineering*, 137:101998.

Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets llm: Can large language models understand structured table data? a benchmark and empirical study. In *Proceedings of the 17th ACM International Conference on Web Search and Data Mining*, WSDM '24, pages 645–654, New York, NY, USA. Association for Computing Machinery.

Anirudh Sundar, Christopher Richardson, and Larry Heck. 2024. gtbls: Generating tables from text by conditional question answering. *arXiv preprint arXiv:2403.14457*.

Lucy Lu Wang, Kyle Lo, Yoganand Chandrasekhar, Russell Reas, Jiangjiang Yang, Douglas Burdick, Darrin Eide, Kathryn Funk, Yannis Katsis, Rodney Kinney, et al. 2020. Cord-19: The covid-19 open research dataset. *ArXiv*, pages arXiv–2004.

Xueqing Wu, Jiacheng Zhang, and Hang Li. 2022. Text-to-table: A new dataset and method for structured table generation. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 2518–2533.

Wael M.S. Yafooz, Siti Z.Z. Abidin, Nasiroh Omar, and Zanariah Idrus. 2013. Managing unstructured data in relational databases. In *2013 IEEE Conference on Systems, Process and Control (ICSPC)*, pages 198–203.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, Qingming Ma, Irene Li, Shanelle Yao, Yi Zhang, et al. 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 (EMNLP)*, pages 3911–3921.

Yuniarti Yuliana and Chia-Hui Chang. 2016. Afis: Automatic format-induction system for detail web pages. In *The Asian Conference on Artificial Intelligence (TAAI)*.

Yuniarti Yuliana and Chia-Hui Chang. 2020. Dcade: Dynamic content alignment for data extraction from web pages. *Journal of Information Science*, 46(5):656–674.

Ce Zhang, Jan Hoffmann, Ce Wang, et al. 2016. Deepdive: Declarative knowledge base construction. *Communications of the ACM*, 60(5):93–102.

Fan Zhang, Alan Ritter, et al. 2019. Openki: Integrating open information extraction and knowledge bases. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics (ACL)*.

## A Definitions

1. **Canonical Join Query:** The canonical join of the database schema is the natural join of all the relations in the schema. (Elmasri and Navathe, 2016)

2. **Primary Key:** A primary key is a set of one or more attributes that uniquely identifies a tuple within a relation. No attribute in the primary key can have a null value. (Silberschatz et al., 2020, Section 3.3.2)

3. **Foreign Key:** A foreign key is an attribute, or a set of attributes, in one relation that references the primary key of another relation. It ensures referential integrity between the two relations. (Silberschatz et al., 2020, Section 3.4)

4. **Referential Integrity:** Referential integrity is a property of a relational database that ensures that every foreign key value in a child table either matches a valid primary key in the referenced parent table or is null (if allowed). It guarantees that relationships between tables remain consistent. (Silberschatz et al., 2020, Section 3.4)

## B Dataset

Our dataset generation approach is illustrated in Fig. 5. For the BIRD dataset, we flatten each multi-table database obtained from the BIRD benchmark (Li et al., 2024) into a single table by joining related tables. Then, using the LLAMA-8B-INSTRUCT model (see prompts in Appendix 6), we generate a natural language sentence for each row. Five consecutive sentences are concatenated to create a paragraph-style input document. The same approach is applied to the Kaggle datasets (Kiattisak, 2023; Rai, 2023; Becker and Kohavi, 1996).

## C Metrics

We assess the quality of both the generated schema and its instantiated content using a suite of novel evaluation metrics that capture structural correctness, semantic alignment, and data fidelity, providing a comprehensive measure of generation quality.

### C.0.1 Schema Evaluation

We evaluate the quality of generated database schemas using three complementary metrics: `Entity Coverage Score (ECS)` for column-level semantic alignment, `Primary Key Coverage (PKC)` for schema completeness, and `Foreign Key Coverage (FKC)` for referential integrity.

**`Entity Coverage Score (ECS)`** evaluates how well the predicted schema recovers the ground truth column names. Let $\{c_1, \ldots, c_N\}$ be the ground truth column names and $\{\hat{c}_1, \ldots, \hat{c}_M\}$ be the predicted columns. For each ground truth column $c_i$, we compute its cosine similarity with every predicted column $\hat{c}_j$ and select the highest similarity. ECS is the average of these maximum scores:

$$\text{ECS} = \frac{1}{N} \sum_{i=1}^{N} \max_{j \in [1,M]} \texttt{cos\_sim}(c_i, \hat{c}_j) \quad (1)$$

where cosine similarity is computed as:

$$\texttt{cos\_sim}(u, v) = \frac{u \cdot v}{\|u\| \|v\|} \quad (2)$$

This metric captures the best semantic match for each ground truth column using SentenceTransformer embeddings (`all-MiniLM-L6-v2`).

**`Primary Key Coverage (PKC)`** measures how well the generated schema supports tuple-level uniqueness by checking whether primary keys are defined. PKC is defined as:

$$\text{PKC} = \frac{\texttt{Num\_PK}}{\texttt{Num\_tables}} \quad (3)$$

Here, `Num_PK` is the number of generated tables that define at least one primary key, and `Num_tables` is the total number of generated tables. This metric reflects the model's ability to generate structurally valid tables that enforce row-level uniqueness through primary keys.

**`Foreign Key Coverage (FKC)`** assesses the extent to which the generated schema maintains referential integrity across tables. FKC is defined as:

$$\text{FKC} = \frac{\texttt{Num\_FK}_{\texttt{Valid}}}{\texttt{Num\_FK}} \quad (4)$$

Here, $\texttt{Num\_FK}_{\texttt{Valid}}$ is the number of foreign keys that correctly reference existing primary keys, and `Num_FK` is the total number of generated foreign keys. This metric evaluates the model's ability to establish valid inter-table relationships, ensuring that foreign keys point to legitimate primary key targets.

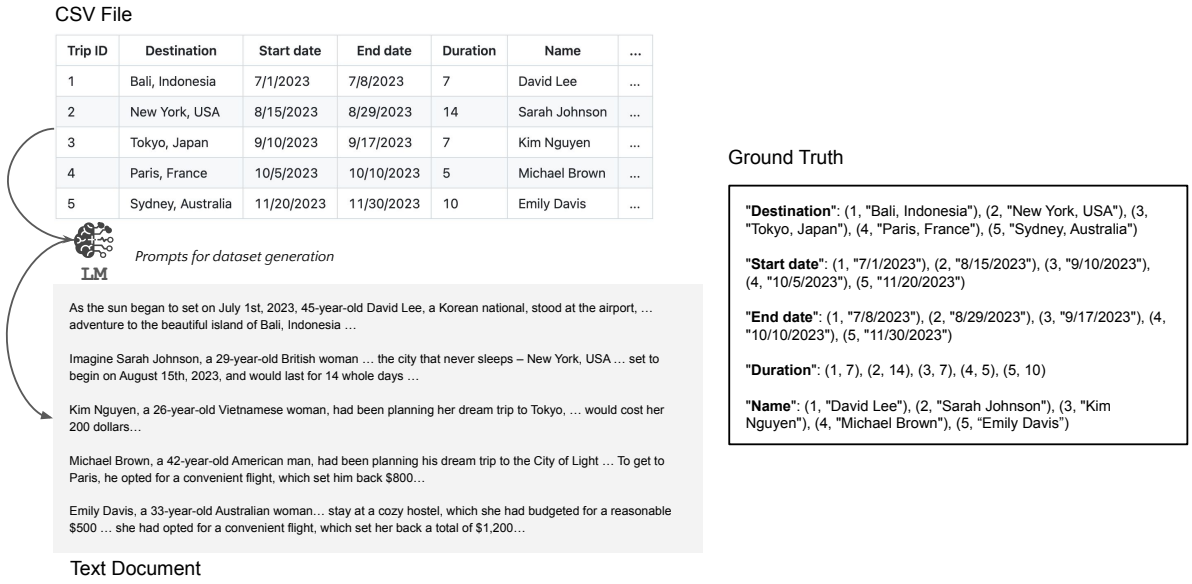


Figure 5: Our dataset generation process

### C.0.2 Database Evaluation

We use five evaluation metrics to assess how well the generated database reconstructs the ground truth data: `Database Construction Success Rate (DBR)`, `Referential Integrity Rate (RRIR)`, `Tuple Coverage (TC)`, `Value Coverage (VC)`, and `Column Consistency (CC)`.

`Database Construction Success Rate (DBR)` captures the percentage of successfully generated databases from text documents.

$$\texttt{DBR} = \frac{\#\texttt{Generated DB}}{\#\texttt{Text Documents}} \tag{5}$$

**`Referential Integrity Rate (RRIR)`** captures whether foreign key joins result in meaningful, non-sparse rows during execution. Let $\mathcal{D}$ be the set of evaluated databases, and let each database $d \in \mathcal{D}$ produce a set of rows $\mathcal{R}_d$ from a canonical foreign key join. For each row $r \in \mathcal{R}_d$, let $n_{\texttt{total}}(r)$ be the number of columns, and $n_{\texttt{null}}(r)$ the number of columns with null values. The per-database score is:

$$\texttt{RRIR}(d) = \frac{1}{|\mathcal{R}_d|} \sum_{r \in \mathcal{R}_d} \left(1 - \frac{n_{\texttt{null}}(r)}{n_{\texttt{total}}(r)}\right) \tag{6}$$

The overall score across all databases is:

$$\texttt{RRIR} = \frac{1}{|\mathcal{D}|} \sum_{d \in \mathcal{D}} \texttt{RRIP}(d) \tag{7}$$

This metric provides a practical signal of referential soundness during execution by quantifying the completeness of joined rows in terms of non-null content.

**`Tuple Coverage (TC)`** quantifies how many ground truth rows are recovered through canonical joins. Let $\mathcal{R}_{\texttt{GT}}$ be the set of primary keys from the ground truth database, and $\mathcal{R}_{\texttt{join}}$ be the set of primary keys resulting from the canonical join query over the generated database. Then:

$$\texttt{TC} = \frac{|\mathcal{R}_{\texttt{GT}} \cap \mathcal{R}_{\texttt{join}}|}{|\mathcal{R}_{\texttt{GT}}|} \tag{8}$$

This metric reflects the row-level reconstruction accuracy.

**`Value Coverage (VC)`** measures the proportion of ground truth cell values that are accurately recovered in the predicted database. A predicted value $\hat{v}$ is considered a match to a ground truth value $v$ if:

- For numeric values: $|v - \hat{v}| < 10^{-2}$ (i.e., absolute difference less than 0.01).
- For textual values: the cosine similarity between embeddings satisfies $\texttt{cos\_sim}(v, \hat{v}) > 0.8$.

Let $\mathcal{V}_{\texttt{GT}}$ be the set of all ground truth values, and $\mathcal{V}_{\texttt{DB}}$ be the set of predicted values matched to ground truth under the criteria above. Then VC is defined as:

$$\text{VC} = \frac{|\mathcal{V}_{\texttt{GT}} \cap \mathcal{V}_{\texttt{DB}}|}{|\mathcal{V}_{\texttt{GT}}|} \tag{9}$$

This ratio reflects the overall proportion of correctly reconstructed cell values, incorporating both numeric precision and semantic similarity for text.

**Column Consistency (CC)** quantifies the proportion of matched values that appear under the correct column names in the predicted database. A column name in the prediction is considered correct if its semantic similarity with the corresponding ground truth column name exceeds a threshold of 0.7, i.e.,

$$\texttt{cos\_sim}(\text{col}_{\texttt{GT}}, \text{col}_{\texttt{DB}}) > 0.7$$

Formally, restricting the sets $\mathcal{V}_{\texttt{GT}}$ and $\mathcal{V}_{\texttt{DB}}$ to values within a specific column `col`, CC is defined as:

$$\text{CC} = \frac{|\mathcal{V}_{\texttt{GT}} \cap \mathcal{V}_{\texttt{DB}}|}{|\mathcal{V}_{\texttt{GT}}|}, \quad \text{where } \mathcal{V}_{\texttt{GT}}, \mathcal{V}_{\texttt{DB}} \in \texttt{col} \tag{10}$$

Here, the intersection counts only those values matched under semantically correct columns according to the cosine similarity criterion above.

# D Experimentation Details

## D.1 Experimental Setting

**Computational Resources and Model Sizes.** We report the number of parameters, computational budget, and infrastructure details for all models and experiments used in this work. The models employed include: LLAMA3-8B-INSTRUCT (8B parameters), CLAUDE 3.7 SONNET (parameter size not disclosed), GPT-4O (parameter size not disclosed), QWEN3-8B (8B parameters), and DEEPSEEK-V2.5 (16B parameters). All experiments, including both development and final evaluation runs, were conducted using 1 GPU (NVIDIA A10, 24 GB VRAM) over a total of approximately 100 GPU hours. Our computing environment included 48-core Intel Xeon Silver 4310 CPUs and 128 GB RAM, running on Ubuntu 24.04.2 LTS. These details are provided to support reproducibility and contextualize the performance reported in this study.

## D.2 Results

We report additional results from our study in this section. Table 8 presents schema coverage scores across different domains and datasets for the DEEPSEEK-V2.5 model and CoT approach. Table 9 shows the impact of different value sources on Tuple Coverage (`TC`), Value Coverage (`VC`) and Column Consistency (`CC`).

| | Domain | ECS | PKC | FKC |
|---|---|---|---|---|
| Kaggle | Tourism | 89.15 | 100 | 100 |
| | Education | **91.08** | 100 | 100 |
| | Finance | 84.71 | 100 | 100 |
| BIRD | California Schools | 76.84 | 100 | 100 |
| | Superhero | 77.55 | 100 | 100 |
| | Books | 84.87 | 100 | 100 |
| | Computer Student | 57.46 | 100 | 100 |
| | Mental Health Survey | **38.97** | 100 | 100 |
| | Authors | 86.44 | 100 | 100 |

Table 8: Schema coverage scores across different domains and datasets for the DEEPSEEK-V2.5 model and CoT approach.

We also conducted a small-scale evaluation on real text from WikiBio (Lebret et al., 2016), CNN-DM (Nallapati et al., 2016), and CORD-19 (Wang et al., 2020) manually annotating 10 entries per dataset to evaluate SQUiD. Results (Tables 10 and 11) align with those from synthetic data, demonstrating SQUiD's robustness. Notably, on WikiBio, GPT-4O fails completely (0% DBR), while SQUiD succeeds in 100% of cases.

## D.3 Additional Context

**Baseline Join Query vs SQUiD Join Query.** For the baseline case, the model was prompted to generate join queries after seeing the full table contents, allowing it to tailor joins to observed values. In contrast, SQUiD 's join queries are issued independently of table population, which may result in more `None` retrievals.

# E Related Work

Recent research relevant to our task of synthesizing relational databases from unstructured text spans three primary areas: (1) summarizing structured information from text (2) interacting with or modifying existing databases (3) domain-specific, non-LLM approaches based on rule-based or statistical methods for relational structure extraction from text.

**Summarizing Structures from Text.** A widely studied area related to our task is *Open Information Extraction* (OpenIE), which extracts subject,

| Model | Diff. | $\mathbb{T}$ (1) | | | $\mathbb{S}$ (2) | | | $\mathbb{L}$ (3) | | | $\mathbb{T}\oplus\mathbb{S}$ (1)+(2) | | | $\mathbb{T}\oplus\mathbb{L}$ (1)+(3) | | | SQUiD (1)+(2)+(3) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | TC | VC | CC | TC | VC | CC | TC | VC | CC | TC | VC | CC | TC | VC | CC | TC | VC | CC |
| CLAUDE 3.7 SONNET | Easy | 1.00 | 0.97 | 0.97 | 1.00 | 0.97 | 0.97 | 0.98 | 0.93 | 0.93 | 1.00 | 0.98 | 0.98 | 1.00 | 0.97 | 0.97 | 1.00 | 0.98 | 0.98 |
| | Med | 0.85 | 0.68 | 0.64 | 0.93 | 0.74 | 0.70 | 0.93 | 0.74 | 0.70 | 0.97 | 0.77 | 0.72 | 0.97 | 0.77 | 0.72 | 0.98 | 0.78 | 0.74 |
| | Hard | 0.82 | 0.51 | 0.33 | 0.94 | 0.58 | 0.37 | 0.88 | 0.51 | 0.33 | 0.96 | 0.60 | 0.40 | 0.97 | 0.60 | 0.39 | 1.00 | 0.63 | 0.41 |
| | Avg. | 0.89 | 0.72 | 0.65 | 0.96 | 0.76 | 0.68 | 0.93 | 0.72 | 0.65 | 0.98 | 0.78 | 0.70 | 0.98 | 0.78 | 0.70 | 0.99 | 0.80 | 0.71 |
| DEEPSEEK-V2.5 | Easy | 0.99 | 0.92 | 0.92 | 0.99 | 0.94 | 0.94 | 1.00 | 0.92 | 0.91 | 1.00 | 0.96 | 0.96 | 1.00 | 0.95 | 0.95 | 1.00 | 0.96 | 0.96 |
| | Med | 0.96 | 0.76 | 0.71 | 0.87 | 0.68 | 0.64 | 0.90 | 0.69 | 0.66 | 0.98 | 0.79 | 0.75 | 0.98 | 0.80 | 0.76 | 0.99 | 0.80 | 0.77 |
| | Hard | 0.89 | 0.54 | 0.35 | 0.83 | 0.42 | 0.26 | 0.75 | 0.35 | 0.24 | 0.95 | 0.57 | 0.38 | 0.95 | 0.57 | 0.38 | 0.95 | 0.59 | 0.39 |
| | Avg. | 0.95 | 0.74 | 0.66 | 0.90 | 0.68 | 0.62 | 0.88 | 0.65 | 0.60 | 0.98 | 0.78 | 0.70 | 0.98 | 0.78 | 0.70 | 0.98 | 0.79 | 0.71 |
| GPT-4O | Easy | 0.98 | 0.90 | 0.90 | 0.98 | 0.93 | 0.93 | 1.00 | 0.90 | 0.89 | 0.99 | 0.95 | 0.95 | 1.00 | 0.96 | 0.96 | 1.00 | 0.97 | 0.97 |
| | Med | 0.94 | 0.75 | 0.71 | 0.88 | 0.69 | 0.66 | 0.90 | 0.68 | 0.64 | 0.98 | 0.80 | 0.76 | 0.98 | 0.81 | 0.77 | 0.99 | 0.81 | 0.77 |
| | Hard | 0.81 | 0.50 | 0.32 | 0.78 | 0.41 | 0.29 | 0.93 | 0.51 | 0.33 | 0.92 | 0.56 | 0.38 | 0.97 | 0.59 | 0.38 | 0.97 | 0.61 | 0.40 |
| | Avg. | 0.91 | 0.71 | 0.64 | 0.88 | 0.67 | 0.63 | 0.94 | 0.69 | 0.62 | 0.97 | 0.77 | 0.70 | 0.98 | 0.79 | 0.70 | 0.99 | 0.80 | 0.71 |
| LLAMA3-8B-INSTRUCT | Easy | 0.99 | 0.89 | 0.88 | 0.81 | 0.74 | 0.74 | 0.77 | 0.61 | 0.60 | 1.00 | 0.95 | 0.95 | 1.00 | 0.94 | 0.94 | 1.00 | 0.95 | 0.95 |
| | Med | 0.95 | 0.70 | 0.66 | 0.76 | 0.52 | 0.48 | 0.89 | 0.61 | 0.57 | 0.98 | 0.75 | 0.72 | 0.98 | 0.76 | 0.73 | 0.99 | 0.79 | 0.75 |
| | Hard | 0.95 | 0.60 | 0.39 | 0.88 | 0.37 | 0.26 | 0.71 | 0.40 | 0.25 | 1.00 | 0.64 | 0.41 | 1.00 | 0.68 | 0.42 | 1.00 | 0.70 | 0.44 |
| | Avg. | 0.96 | 0.73 | 0.64 | 0.81 | 0.54 | 0.49 | 0.79 | 0.54 | 0.47 | 0.99 | 0.78 | 0.69 | 0.99 | 0.79 | 0.70 | 1.00 | 0.81 | 0.71 |
| QWEN3-8B | Easy | 0.99 | 0.92 | 0.92 | 0.97 | 0.92 | 0.92 | 0.85 | 0.72 | 0.72 | 1.00 | 0.96 | 0.96 | 1.00 | 0.96 | 0.96 | 1.00 | 0.96 | 0.96 |
| | Med | 0.94 | 0.71 | 0.71 | 0.90 | 0.71 | 0.71 | 0.96 | 0.67 | 0.67 | 0.95 | 0.74 | 0.73 | 0.98 | 0.78 | 0.78 | 0.98 | 0.79 | 0.79 |
| | Hard | 0.57 | 0.29 | 0.29 | 0.36 | 0.23 | 0.23 | 0.76 | 0.35 | 0.35 | 0.59 | 0.33 | 0.33 | 0.99 | 0.48 | 0.48 | 0.99 | 0.51 | 0.51 |
| | Avg. | 0.83 | 0.64 | 0.64 | 0.75 | 0.62 | 0.62 | 0.85 | 0.58 | 0.58 | 0.85 | 0.68 | 0.67 | 0.99 | 0.74 | 0.74 | 0.99 | 0.76 | 0.75 |

Table 9: Impact of different value sources on Tuple Coverage (TC), Value Coverage (VC) and Column Consistency (CC). $\mathbb{T}\oplus\mathbb{S}$ combines tuples generated from $\mathbb{T}$ and $\mathbb{S}$ while $\mathbb{T}\oplus\mathbb{L}$ combines $\mathbb{T}$ and $\mathbb{L}$. SQUiD combines outputs from all three prompts.

| Model | Dataset | ECS | PKC | FKC |
|---|---|---|---|---|
| DEEPSEEK-V2.5 | WikiBio | 87.2 | 100 | 100 |
| | CNN-DM | 75.2 | 100 | 100 |
| | CORD-19 | 44.3 | 100 | 100 |
| LLAMA-3-8B-INSTRUCT | WikiBio | 78.2 | 100 | 100 |
| | CNN-DM | 63.6 | 100 | 100 |
| | CORD-19 | 60.6 | 100 | 100 |
| GPT-4O | WikiBio | 81.1 | 100 | 100 |
| | CNN-DM | 67.1 | 100 | 100 |
| | CORD-19 | 64.1 | 100 | 100 |

Table 10: Schema evaluation across real text datasets. Results for schema generation failures that violate the requested structure are omitted.

predicate, object (SPO) triplets from unstructured text (Niklaus et al., 2018). While OpenIE provides useful abstractions, the extracted triplets are not organized under a formal data model. A more structured alternative is the *text-to-table* generation task. Early works approach this as a sequence modeling problem, jointly generating column headers and cell contents (Wu et al., 2022). More recent systems such as **gTBLS**(Sundar et al., 2024) and **Seq2Seq&Set**(Li et al., 2023) decouple schema inference from data population in the *text-to-table* task, yielding improvements in table validity and structure. Other lines of work explore extracting structured data from semi-structured documents such as HTML and PDF using LLMs (Arora et al., 2023), or schema-driven information extraction from heterogeneous tables (Bai et al., 2023). However, the outputs remain flat and lack the normalized relationships central to relational database design. **T3** (Deng et al., 2024) takes a step further by converting extracted tuples into flat tables, which is conceptually closest to our use of intermediate triplet representations. Still, their method does not capture inter-table relationships, limiting alignment with relational database requirements. Additionally, other research explores non-relational structures such as mind maps for representing extracted information (Jain et al., 2024), which similarly do not align with the relational database model our work targets.

**Manipulating Existing Databases.** Another line of work focuses on interacting with or updating existing relational databases using language. Early work such as (Mansuri and Sarawagi, 2006) proposed integrating unstructured sources into relational databases using information extraction and matching techniques, but relied heavily on statistical models, rule-based systems and domain-specific heuristics. In **TEXT2DB** (Jiao et al., 2024), LLM agents ingest documents and update a pre-existing relational database. While it operates on relational databases, it assumes an existing database with a predefined schema and does not attempt to synthesize a new one. On the other hand, the *text-to-SQL* literature (Hong et al., 2024; Yu et al., 2018) focuses on translating natural language queries into

| Model | Dataset | DBR (SQ×) | TC (SQ×) | VC (SQ×) | CC (SQ×) | DBR | TC | VC | CC |
|---|---|---|---|---|---|---|---|---|---|
| CLAUDE 3.7 SONNET | WikiBio | 100 (2.0×) | 100 (3.2×) | 91.2 (4.8×) | 87.5 (4.6×) | 49.2 | 31.2 | 19.2 | 19.2 |
| | CNN-DM | 100 (4.0×) | 100 (5.8×) | 66.0 (7.4×) | 63.2 (7.1×) | 25.0 | 17.2 | 8.9 | 8.9 |
| | CORD-19 | 100 (4.0×) | 100 (8.0×) | 68.3 (10.7×) | 56.1 (8.8×) | 25.0 | 12.5 | 6.4 | 6.4 |
| DEEPSEEK-V2.5 | WikiBio | 100 (2.0×) | 100 (2.4×) | 93.0 (9.1×) | 89.3 (9.9×) | 51.0 | 41.0 | 10.2 | 9.0 |
| | CNN-DM | 100 (3.6×) | 100 (4.6×) | 65.0 (5.7×) | 65.0 (5.7×) | 28.0 | 21.7 | 11.5 | 11.5 |
| | CORD-19 | 100 (5.4×) | 100 (6.2×) | 76.1 (9.3×) | 74.2 (9.0×) | 18.5 | 16.2 | 8.2 | 8.2 |
| LLAMA-3-8B | WikiBio | 100 (5×) | 100 (5×) | 92.7 (5.1×) | 85.7 (4.7×) | 20.0 | 20.0 | 18.1 | 18.1 |
| | CNN-DM | 100 (1.7×) | 100 (2.7×) | 74.2 (4.1×) | 71.8 (4.0×) | 60.0 | 37.7 | 18.2 | 18.2 |
| | CORD-19 | 100 (2×) | 100 (2×) | 73.3 (4.4×) | 73.3 (4.4×) | 50.0 | 50.7 | 16.5 | 16.5 |
| GPT-4O | WikiBio | 100 (∞) | 100 (∞) | 94.6 (∞) | 92.4 (∞) | 0 | 0 | 0 | 0 |
| | CNN-DM | 100 (3.3×) | 100 (5.2×) | 77.8 (11.8×) | 74.6 (11.3×) | 30.0 | 19.2 | 6.6 | 6.6 |
| | CORD-19 | 100 (10×) | 100 (12.2×) | 79.5 (19.9×) | 77.3 (19.3×) | 10.0 | 8.22 | 4.0 | 4.0 |

Table 11: SQUID vs. baseline on real datasets. Metrics: DBR = Database Construction Rate, TC = Tuple Coverage, VC = Value Coverage, CC = Column Consistency. (SQ×) = SQUiD's multiplier over baseline.

executable SQL statements over a known schema. Other works in this space include relation-aware schema encoding for better generalization (Pang et al., 2020), constrained decoding for syntactically valid SQL generation (Scholak et al., 2021), and synthetic data generation to improve model robustness (Chang et al., 2024). However, none of these works attempt to synthesize a relational database from text.

**Non-LLM Approaches for Relational Structure Extraction From Text.** Before LLMs, integrating unstructured text into relational databases relied on classical pipelines combining information extraction, schema induction, and entity linking. Systems such as **DeepDive** (Zhang et al., 2016), **LILLIE** (Smith et al., 2022b), and **OpenKI** (Zhang et al., 2019) extracted structured facts and aligned them with relational schemas using statistical inference, symbolic reasoning, or context-aware matching. In web-centric domains, methods like **SEDE** (Deng, 2010, 2011) and wrapper induction systems (Carlson and Schafer, 2008; Chang and Wu, 2016; Yuliana and Chang, 2016, 2020) inferred schemas from repeated HTML patterns and populated tables using DOM-based alignment. Statistical models such as **SICTF** (Nimishakavi and Talukdar, 2016) induced relation schemas from OpenIE triples via joint tensor factorization. These non-LLM methods demonstrated the feasibility of relational synthesis via symbolic or statistical reasoning, but typically required domain-specific tuning and struggled to generalize across diverse, noisy input text.

Broadly, existing research either aims to extract tables from text or to interface with predefined relational databases—without bridging the gap between the two. To our knowledge, no existing work performs fully automated and domain-generalized *text-to-relational database synthesis*. Our system fills this gap by leveraging a neurosymbolic framework that decomposes the task into interpretable stages.

# F Artifact Use

## F.1 Dataset License Information

In accordance with ACL guidelines, we disclose the licenses of all datasets used.

The BIRD benchmark datasets (Li et al., 2024) are distributed under various open licenses including Public Domain, CC0, CC-BY 4.0, CC-BY-SA 4.0, GPL, and CPOL, all permitting research use and redistribution.

The Kaggle datasets utilized in our experiments are licensed as follows, and all allow research use and redistribution:

- **Tourism dataset** (Kiattisak, 2023): Licensed under Creative Commons Attribution 4.0 (CC-BY 4.0).
- **Education dataset** (Becker and Kohavi, 1996): Licensed under Creative Commons Attribution 4.0 (CC-BY 4.0).
- **Finance dataset** (Rai, 2023): Licensed under the MIT License.

Additionally, we will release our generated dataset publicly under a CC BY 4.0 License.

## F.2 Software and Language Models

We used Stanford CoreNLP (v4.5.9) (Manning et al., 2014), licensed under GNU GPLv3, which permits free use, modification, and redistribution under open-source terms.

The language models employed are publicly available and used under their respective license or terms of service:

- **LLaMa-3-8B-Instruct** (Meta AI, 2024): Released under Meta's research license allowing academic use.
- **Claude 3.7 Sonnet** (Anthropic, 2024): Provided under Anthropic's terms for research and commercial use.
- **GPT-4o** (OpenAI, 2024): Accessed via OpenAI's API under their usage policies.
- **Qwen3-8B** (Alibaba, 2024): Released with a permissive license for research use.
- **DeepSeek-V2.5** (DeepSeek AI, 2024): Licensed for research use as specified by DeepSeek AI.

## G Prompts

All of the prompts we use in SQUiD are provided in Figures 6 to 20.

```
You are a creative AI that rephrases given sentences into engaging, conversational stories while
incorporating all provided datapoints.

- Ensure that no information is omitted or added, and skip any datapoints labeled as 'nan'.
- Do not rephrase the object of a sentence. For example, if the sentence is 'start date is $9/22/2023
$', do not change the date to a different format.
- Respond only with the rephrased sentence without any additional commentary.
```

Figure 6: Prompts for dataset generation with LLAMA3-8B-INSTRUCT: system prompt

```
Rephrase the following sentence into a conversational story, ensuring all data points are included
while skipping 'nan' values.
Do not introduce any extra or false details.

Original sentence: {sentence}

Creative sentence:
```

Figure 7: Prompts for dataset generation with LLAMA3-8B-INSTRUCT: user prompt template

```
You are an expert at formulating database schemas from textual data. I have given you a paragraph of
text.

Using this text, your task is to generate a relational database schema in JSON format.

### **Task:**
1. **Extract Entities & Relationships**: Identify unique entity types and relationships.
2. **Determine Attributes**: Define necessary columns for each table.
3. **Normalize the Schema**: Ensure proper **primary keys, foreign keys, and normalization (3NF)**.
4. **Generate Output in JSON Format**.
5. **Column and Table name restriction**:

reserved_sql_keywords = ["order", "group", "select", "from", "where", "join", "on", "as", "and", "or
", "by", "insert", "update", "delete", "create", "drop", "alter", "into", "table"]
- Ensure that the table names and column names do not contain any SQL reserved keywords.
```

Figure 8: Prompts for schema generation: system prompt

```
### **Text:**
{text}

### **Expected Example Output Format (Strictly Follow This Structure while modifying the table_names,
 column_names to match the given text)**:
schema = [
    {{
        "table_name": "student",
        "columns": [
            {{"name": "id", "type": "INTEGER", "primary_key": True}},
            {{"name": "name", "type": "TEXT"}},
        ]
    }},
    {{
        "table_name": "course",
        "columns": [
            {{"name": "id", "type": "INTEGER", "primary_key": True}},
            {{"name": "title", "type": "TEXT"}},
        ]
    }},
    {{
        "table_name": "enrollment",
        "columns": [
            {{"name": "id", "type": "INTEGER", "primary_key": True}},
            {{"name": "student_id", "type": "INTEGER", "foreign_key": True, "foreign_key_table": "
student", "foreign_key_column": "id"}},
            {{"name": "course_id", "type": "INTEGER", "foreign_key": True, "foreign_key_table": "
course", "foreign_key_column": "id"}}
        ]
    }}
]


Now output the schema as per the system instructions.
### Output:
```

Figure 9: Prompts for schema generation: user prompt template

```
You are an expert at formulating database schemas from textual data. I have given you a paragraph of
text.
Using this text, your task is to generate a relational database schema in JSON format.

---

### **Step-by-Step Guide for Schema Creation (Follow This Chain of Thought)**

**Requirements Analysis**:
- Identify all distinct entities and attributes from the text.
- Determine necessary tables and their columns.

**Entity-Relationship (ER) Modeling**:
- Identify entity relationships (One-to-One, One-to-Many, Many-to-Many).
- If applicable, use associative tables for Many-to-Many relationships.

**Define Tables and Columns**:
- Convert entities into relational tables with appropriate **data types**.

**Establish Primary Keys**:
- Assign a **Primary Key (PK)** for each table to uniquely identify records.

**Define Relationships and Foreign Keys**:
- Use **Foreign Keys (FK)** to enforce referential integrity between tables.
- Ensure that it is possible to join all tables to create one flat table using the foreign keys.
- Apply **ON DELETE CASCADE** if necessary to maintain consistency.

**Normalization (1NF => 2NF => 3NF)**:
- Ensure atomic values (1NF).
- Remove partial dependencies (2NF).
- Eliminate transitive dependencies (3NF).

**Define Constraints**:
- Apply **NOT NULL**, **UNIQUE**, **CHECK**, and other constraints as needed.

**Indexing for Performance**:
- Create indexes on frequently queried columns (e.g., search fields).

- **Column and Table name restriction**:
reserved_sql_keywords = ["order", "group", "select", "from", "where", "join", "on", "as", "and", "or
",
        "by", "insert", "update", "delete", "create", "drop", "alter", "into", "table"]
- Ensure that the table names and column names do not contain any SQL reserved keywords.

---
### **Task Instructions:**
- **Step through the schema creation process using the above guide**.
- **Generate a well-structured, normalized relational database schema**.
- **Output only the final schema** in Python dictionary format (NO explanations).
- **Column and Table name restriction**:
reserved_sql_keywords = ["order", "group", "select", "from", "where", "join", "on", "as", "and", "or
",
        "by", "insert", "update", "delete", "create", "drop", "alter", "into", "table"]
- Ensure that the table names and column names do not contain any SQL reserved keywords.
```

Figure 10: Prompts for schema generation: CoT - system prompt

```
### **Text:**
{text}

### **Expected Example Output Format (Strictly Follow This Structure while modifying the table_names,
 column_names to match the given text)**:
schema = [
    {{
        "table_name": "student",
        "columns": [
            {{"name": "id", "type": "INTEGER", "primary_key": True}},
            {{"name": "name", "type": "TEXT"}},
        ]
    }},
    {{
        "table_name": "course",
        "columns": [
            {{"name": "id", "type": "INTEGER", "primary_key": True}},
            {{"name": "title", "type": "TEXT"}},
        ]
    }},
    {{
        "table_name": "enrollment",
        "columns": [
            {{"name": "id", "type": "INTEGER", "primary_key": True}},
            {{"name": "student_id", "type": "INTEGER", "foreign_key": True, "foreign_key_table": "
student", "foreign_key_column": "id"}},
            {{"name": "course_id", "type": "INTEGER", "foreign_key": True, "foreign_key_table": "
course", "foreign_key_column": "id"}}
        ]
    }}
]

Now output the schema as per the system instructions.
### Output:
```

Figure 11: Prompts for schema generation: CoT - user prompt template

```
You are a helpful assistant that who assists a user with information extraction tasks.
Your job is to associate a unique superkey value with each paragraph in the text.
You will be given multiple paragraphs of text, a database schema, and a superkey.
Your task is to associate the superkey value with each paragraph in the text. Each paragraph MUST be
associated with a superkey value. No two superkey values should be the same.
Fill in the <FILL IN WITH APPROPRIATE VALUE OF {superkey}> with the value. You will not provide code
or SQL, you will do the task yourself.
```

Figure 12: Prompts for triplet generation with LLM- unique identifier association: system prompt

```
**Text**:
{text}

**Schema**:
{schema}

**Superkey**:
{superkey}

**Paragraphs**:
"""
    user_prompt += "\n--\n"
    for j in range(len(paragraphs)):
        user_prompt += f"paragraph {j}: {paragraphs[j]}\n"
        user_prompt += f"associated superkey: <FILL IN WITH APPROPRIATE VALUE OF {superkey}>\n"
        user_prompt += "\n--\n"
```

Figure 13: Prompts for triplet generation with LLM- unique identifier association: user prompt template

```
You are an expert in Open Information Extraction and relational databases. Given a database schema
and a natural language paragraph, your task is to extract all factual information from each sentence
of the paragraph in the form of triplets, structured as a Python list of dictionaries.

Each dictionary should have the following keys: 'table_name', 'column_name', and 'value'. Ensure that
 the extracted triplets strictly follow the format: {'table_name': <table_name>, 'column_name': <
column_name>, 'value': }.

Only extract values that explicitly appear in the input sentence. The table_name and column_name must
 match the schema. Do not invent values or infer unstated facts. You don"t need to generate triplets
for values that are not mentioned. DO NOT generate code, do the task yourself.
```

Figure 14: Prompts for triplet generation with LLM- triplet generation: system prompt

```
You will be given a database schema and a sentence. Extract all relevant triplets of the form:
{{"table_name": <table_name>, "column_name": <column_name>, "value": <value>}}
Your output must be a valid Python list of dictionaries. Do not include any explanations or notes-
only return the list.
{example_schema}

Sentence: {example_text}

{example_output}

Now extract triplets for the following input:

Schema:
{schema}

Sentence: {text}
Triplets:
```

Figure 15: Prompts for triplet generation with LLM- triplet generation: user prompt template

```
Extract information using the extraction tool -- "extract".

Example:
Python table schema:
    - person: "id: int [PK]", "name: string", "age: int", "location: string", "married: boolean"
    - job: "id: int [PK]", "person_id: int [FK => person(id)]", "title: string", "salary: int", "
department: string"


Example triplets:
    < 1 , person, name, John >
    < 1 , person, age, 30 >
    < 1 , person, location, Los Angeles >
    < 1 , person, married, false >
    < 1 , job, title, Software Engineer >
    < 1 , job, salary, 100000 >
    < 1 , job, department, Engineering >

Example usage:
    extract person: "id": 1; "name": "John"; "age" 30; "location": "Los Angeles"; "married": false
    extract job: "id": 1; "person_id": 1; "title": "Software Engineer"; "salary": 100000; "department
": "Engineering"

Tool guidelines:
- "id" should always be present in the extraction. It is the primary key of the table. You must
assign an appropriate value to it. Do not leave it empty.
- "xxx_id" is the foreign key of the table and references the primary key of a row in another table.
It must also be present in the extraction, you must assign an appropriate value to it, and ensure
that the value matches the id of the row in it's foreign key parent table. Do not leave it empty.
- You must follow the order of the columns per table, defined in the Python table schema.
- You must follow the data type for each column.
- You must use "extract" once per table of the schema to extract information from the whole paragraph
 and the set of triplets for each of the table.
- After reading the paragraph, the schema and the triplets, do all the extraction steps in one go, do
 not skip any extraction steps. Do not repeat extraction for the same id if there are no new
information.
- Do not extract any information that is not present in the original paragraph.
- Other than "id" and "xxx_id", if no value is found for a column, use '?' to represent the value. If
 all columns are empty, do not use extract for that row.

Python table schema:
{table_instruction_str}
```

Figure 16: Prompts for table population: tooluse prompt for extraction

```
system_prompt = f"""You are an expert at populating values in a database from text based on a given
database schema. I have provided a paragraph of text and a database schema. Using this information,
your task is to extract relevant values and format them according to the schema. Do not provide code,
 do the task yourself."""
    user_prompt = f"""
### **Text:**
{text}
### **Schema:**
{schema}
### **Expected Output Format:**
{data_template}
Output as many rows as necessary to populate the data. Replace the '#' with the actual values from
the text.

You will follow this chain-of-thought reasoning to generate the final output:
- Generate output entries relevant to the text.
- Follow the given output format strictly. Do not add any additional explanations or comments. Only
output the data entries in given format. Do not provide code, do the task yourself.
### **Output:**
"""
```

Figure 17: Prompts for table population: Method $\mathbb{T}$

```
text = data['text']
schema = data['schema']
superkey = data['superkey']
data_template = generate_empty_data_template_tooluse(schema)
identified_values = data['identified_values']

system_prompt = f"""You are a database assistant that extracts data from each sentence of a given
text, and populates data entries into a fixed relational database schema.

The user will give you the following inputs: a text with data of multiple users, the primary
identifier (superkey) of that text, extracted triplets from that text in the following format:
<superkey, `subject`, `relation`, `object`>

You will follow this chain-of-thought reasoning to generate the final output:
- Generate output entries as per given instructions, relevant to the current paragraph.
- Validate each output entry by going through the triplets one by one, and ensure every unique data
point from the triplets is captured into the correct table and fields.
- Follow the given output format strictly. Do not add any additional explanations or comments. Only
output the data entries in given format. Do not provide code, do the task yourself.

Text:
{text}

Output format:
{data_template}
```

Figure 18: Prompts for table population: Method $\mathbb{S}$

```
text = data['text']
schema = data['schema']
superkey = data['superkey']
data_template = generate_empty_data_template_tooluse(schema)
identified_values = data['identified_values']

system_prompt = f"""You are a database assistant that extracts data from each sentence of a given
text, and populates data entries into a fixed relational database schema.

The user will give you the following inputs: a text with data of multiple users, the primary
identifier (superkey) of that text, extracted triplets from that text in the following format:
<superkey, `table_name`, `column_name`, `value`>

You will follow this chain-of-thought reasoning to generate the final output:
- Generate output entries as per given instructions, relevant to the current paragraph.
- Validate each output entry by going through the triplets one by one, and ensure every unique data
point from the triplets is captured into the correct table and fields.
- Follow the given output format strictly. Do not add any additional explanations or comments. Only
output the data entries in given format. Do not provide code, do the task yourself.

Text:
{text}

Output format:
{data_template}

"""
    user_prompt = "Please perform the task as per the system instructions.\n"
    user_prompt += f"### Extracted <{superkey}, `table_name`, `column_name`, `value`> triplets from
each paragraph:\n"
    for item in identified_values:
        for triplet in item['triplets']:
            if triplet['value'] !="not mentioned" or triplet['value'] !="not provided":
                user_prompt += f"<{item['superkey']}, {triplet['table_name']}, {triplet['column_name
']}, {triplet['value']}>\n"
        user_prompt += "\n"
    user_prompt += "### Output:\n"
```

Figure 19: Prompts for table population: Method $\mathbb{L}$

```
system_prompt = """You are a database expert. Your task is generating SQL 'CREATE TABLE' and 'INSERT
INTO' statements from text."""
    user_prompt = f"""
### **Text:**
{text}
### **Output: (Write the create table and insert into statements together)
"""
```

```
system_prompt = """You are a database expert. Your task is generating a sqlite join query from '
create table' and 'insert into' statements."""
    user_prompt = f"""
### **'CREATE TABLE' and 'INSERT INTO' statements:**
{sql_statements}
### **Output:**
"""
```

Figure 20: Prompts for baseline