# DeKeyNLU: A Dataset for Enhancing Natural Language to SQL Generation through Task Decomposition and Keyword Extraction

Anonymous ACL submission

## Abstract

Natural Language to SQL (NL2SQL) provides a new model-centric paradigm that simplifies database access for non-technical users by converting natural language queries into SQL commands. Recent advancements, particularly those integrating Retrieval-Augmented Generation (RAG) and Chain-of-Thought (CoT) reasoning, have made significant strides in enhancing NL2SQL performance. However, challenges such as inaccurate task decomposition and keyword extraction by LLMs remain major bottlenecks, often leading to errors in SQL generation. While existing datasets aim to mitigate these issues by fine-tuning models, they struggled with over-fragmentation of tasks and lack of domain-specific keyword annotations, limiting their effectiveness. To address these limitations, we present DeKeyNLU, a novel dataset which contains 1,500 meticulously annotated QA pairs aimed at refining task decomposition and enhancing keyword extraction precision for RAG pipline. Fine-tuned with DeKeyNLU, we propose DeKeySQL, a RAG-based NL2SQL pipeline that employs three distinct modules for user question understanding, entity retrieval, and generation to improve SQL generation accuracy. We benchmarked multiple model configurations within DeKeySQL RAG pipeline. Experimental results demonstrate that fine-tuning with DeKeyNLU significantly improves SQL generation accuracy on both BIRD (62.31% to 69.10%) and Spider (84.2% to 88.7%) dev datasets.

## 1 Introduction

The rapidly evolving landscape of data accessibility has intensified the need for intuitive interfaces that empower non-technical users to interact with complex databases [Javaid et al., 2023, Al Naqbi et al., 2024]. Natural Language to SQL (NL2SQL) systems fulfill this requirement by translating user-friendly natural language queries into precise SQL commands, facilitating seamless information retrieval without requiring users to possess programming skills for database question answering (Database QA) [Gao et al., 2023, Hong et al., 2024, Liu et al., 2024].

Despite considerable advancements in NL2SQL methods, accuracy remains a persistent challenge. Modern hybrid approaches, which integrate Chain of Thought (CoT) [Wei et al., 2022] reasoning and Retrieval-Augmented Generation (RAG) [Lewis et al., 2020] with specialized modules—such as CHASE-SQL [Pourreza et al., 2024], CHESS [Talaei et al., 2024], PURPLE [Ren et al., 2024], DTS-SQL [Pourreza and Rafiei, 2024], and MAC-SQL [Wang et al., 2023]—have made significant strides but continue to encounter two major obstacles: inadequate task decomposition and imprecise keyword extraction from user queries. These issues frequently result in logical errors and incorrect field identifications, particularly when queries involve complex, multi-table relationships.

Prior work in the field has attempted to mitigate these challenges. For instance, QDecomp [Tai et al., 2023] and QPL [Eyal et al., 2023] focus on refining query decomposition techniques by prompting with LLMs, while DARA [Fang et al., 2024] strives to enhance natural language understanding (NLU) through agent frameworks. However, these approaches often lead to over-fragmentation of tasks and do not adequately assess the overall model performance in Database QA contexts. Furthermore, fine-tuning existing models with domain-specific data has shown promise, but prevalent datasets, such as BREAK [Wolfson et al., 2020], lack comprehensive domain-specific annotations for Database QA evaluation and do not emphasize the precise keyword extraction required for database retrieval.

To address these gaps, we present DeKeyNLU, a novel dataset specifically designed to enhance NLU capabilities for NL2SQL systems. DeKeyNLU

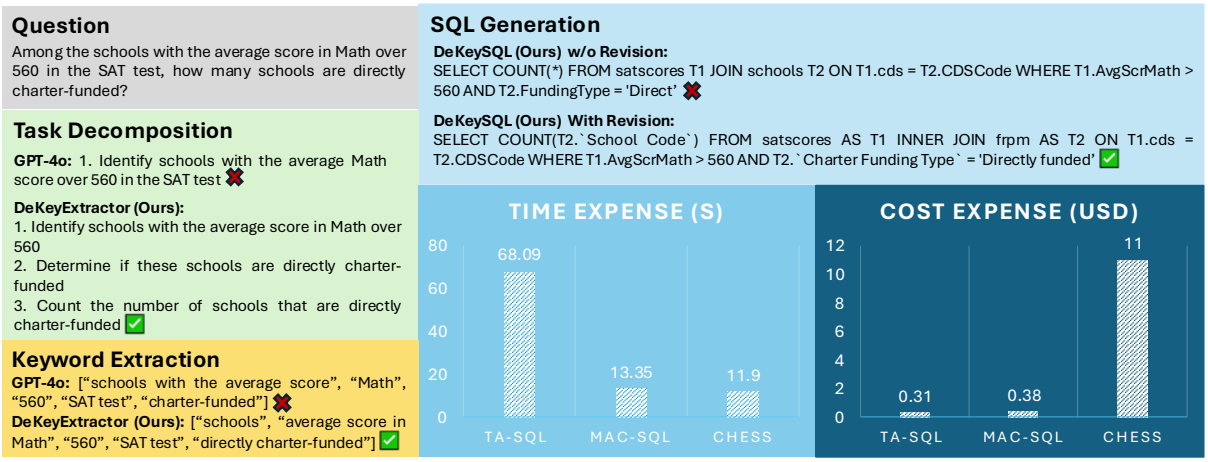


Figure 1: Comparison of advanced NL2SQL methods with DeKeySQL. GPT-4o suffers from incomplete task decomposition and incorrect keyword extraction. Missing a revision module, GPT-4o shows lower code generation accuracy. Methods like MAC-SQL, CHESS, TA-SQL are efficient in either time or cost, but not both.

consists of 1,500 QA pairs meticulously annotated with a focus on two critical aspects: task decomposition and keyword extraction. Originating from the BIRD dataset [Li et al., 2024d], it provides a high-quality benchmark for evaluating and improving NL2SQL methods in Database QA.

In addition, we introduce DeKeySQL, a RAG-based pipeline optimized for NL2SQL tasks. DeKeySQL comprises three key modules: (1) User Question Understanding (UQU), which leverages the DeKeyNLU dataset for task decomposition and keyword extraction; (2) Entity Retrieval, incorporating retrieval and re-ranking to identify database elements relevant to the users' question; and (3) Generation, featuring task reasoning and feedback-driven error correction to produce accurate SQL statements.

We benchmarked multiple model configurations within DeKeySQL RAG-based pipeline. Fine-tuning the UQU module with DeKeyNLU improved SQL generation accuracy from 62.31% to 69.10% on the BIRD dev dataset and from 84.2% to 88.7% on the Spider [Yu et al., 2018] dev dataset. Our experiments reveal that larger models, like GPT-4o-mini[OpenAI, 2024a], excel at task decomposition, while smaller models, such as Mistral-7B [Jiang et al., 2023], are more effective for keyword extraction. We also observed that optimal performance varies depending on dataset size and model architecture. Moreover, across the pipeline components, user question understanding emerged as the most significant factor influencing overall SQL generation accuracy, followed by entity retrieval and revision mechanisms.

## 2 Related Work

### 2.1 Database Question Answering

Database Question Answering (Database QA) aims to provide precise answers derived from tabular data through advanced reasoning. Early research focused on discrete reasoning [Jin et al., 2022], with works such as TAT-QA [Zhu et al., 2021], FinQA [Chen et al., 2021], and MVGE [Ma et al., 2017] exploring methods like fine-tuning, pre-training, and in-context learning. Although these approaches advanced the field, they often struggled with generalization in multi-table settings [Zhang et al., 2024].

Parallelly, NL2SQL methods enable mapping natural language questions to SQL queries, offering efficient solutions [Gao et al., 2023]. This area spans rule-based, neural network-based, pre-trained language model (PLM)-based, and large language model (LLM)-based strategies [Li et al., 2024a]. Rule-based systems [Katsogiannis-Meimarakis and Koutrika, 2021] were gradually superseded by neural and transformer-based methods, such as BERT [Devlin, 2018], which improved performance on benchmarks like ScienceBenchmark [Zhang et al., 2023]. Recent advances leverage LLMs (e.g., GPT-4 [Achiam et al., 2023]), empowering systems such as CHESS [Talaei et al., 2024], DAIL-SQL [Gao et al., 2023], and MAC-SQL [Wang et al., 2023] with specialized modules for enhanced accuracy and output refinement. Nonetheless, LLM-based approaches continue to face challenges like limited accuracy, high resource costs, and runtime constraints, impeding their prac-

ticality [Li et al., 2024a].

### 2.2 Natural Language Understanding for NL2SQL

Natural Language Understanding (NLU) is central to enabling machines to interpret human language [Allen, 1988], supporting tasks from keyword extraction to complex question answering [Yu et al., 2023]. The development of LLMs, including Gemini-Pro [Reid et al., 2024], GPT-4 [Achiam et al., 2023], and Mistral [Jiang et al., 2023], has significantly advanced NLU performance. To further augment their abilities, techniques such as advanced text alignment [Zha et al., 2024], human-provided explanations [Liu et al., 2021], and explicit reasoning frameworks like Chain-of-Thought (CoT) [Wei et al., 2022] and Tree-of-Thought [Yao et al., 2024] are widely studied. Robustness and generalization are assessed via benchmarks such as Adversarial NLI [Nie et al., 2019], OTTA [Deriu et al., 2020], and SemEval-2024 Task 2 [Jullien et al., 2024].

In Database QA and NL2SQL, robust NLU is essential for query understanding and decomposition. Methods like QDecomp [Tai et al., 2023] and QPL [Eyal et al., 2023] break down complex user questions, while DARA [Fang et al., 2024] and Iterated Decomposition [Reppert et al., 2023] iteratively refine intent understanding. Grammar-based models (e.g., IRNet [Guo et al., 2019]) and techniques such as ValueNet [Brunner and Stockinger, 2021] help align natural language with structured schema elements. Yet, accurately handling nuanced queries remains challenging. While datasets like BREAK [Wolfson et al., 2020] support research in decomposition, many suffer from over-segmentation, and traditional NLU datasets fail to capture key Database QA aspects like mapping keywords to database elements. Thus, there is a need for more holistic datasets and evaluation protocols that reflect the real-world requirements of Database QA systems.

## 3 DeKeyNLU Dataset Creation

As illustrated in Figure 1, current LLMs exhibit limitations in NLU capabilities for Database QA, which adversely affects NL2SQL accuracy. Existing datasets like BREAK [Wolfson et al., 2020], while useful, often lack data directly pertinent to the primary task of complex SQL generation. Moreover, their keyword extraction is often not comprehensive or noise-free enough for robust NLU performance evaluation in Database QA. To address these challenges, we developed the DeKeyNLU dataset.

### 3.1 Data Sources

DeKeyNLU is derived from the BIRD dataset [Li et al., 2024d], chosen for its validated origins, large scale, and extensive use in NL2SQL research. BIRD contains 12,751 text-to-SQL pairs across 95 databases (33.4 GB), spanning 37 professional domains, and is specifically designed for evaluating and training NL2SQL models. It integrates 80 open-source relational databases from platforms like Kaggle[1] and Relation.vit. An additional 15 relational databases were created for a hidden test set to prevent data leakage. The BIRD team utilized crowdsourcing to collect natural language questions paired with their corresponding SQL queries.

### 3.2 Selection and Annotation

We randomly selected 1,500 instances from the BIRD training dataset. Each instance consists of a user question and its ground truth SQL query. Our annotation process, depicted in Figure 2, began with initial task decomposition and keyword extraction performed by GPT-4o [OpenAI, 2024b].

Task decomposition involved breaking user questions into a *main task* (primary goal) and *sub-tasks* (refinements of the main task). Keyword extraction categorized terms into *object* (related to table/column names) and *implementation* (filtering criteria, represented as a dictionary of actions and conditions). These elements aid similarity matching within the database.

Despite using CoT [Wei et al., 2022] and few-shot techniques [Brown, 2020], GPT-4o's initial interpretations were often suboptimal, producing redundant/incomplete tasks or incorrect keywords (see left panel of Figure 2). This necessitated manual refinement. Three expert annotators were engaged to review and correct GPT-4o's outputs. A three-phase cyclic process ensured cross-validation: annotators started with different subsets (A, B, C), then exchanged and reviewed, ensuring each instance was evaluated by all. The process involved:

**1. Evaluate Task Decomposition:** Annotators manually assessed the logical consistency of GPT-4o-generated main tasks and sub-tasks, removing redundancies and adding missing relevant tasks.

**2. Evaluate Keyword Extraction:** Keywords

[1] https://www.kaggle.com

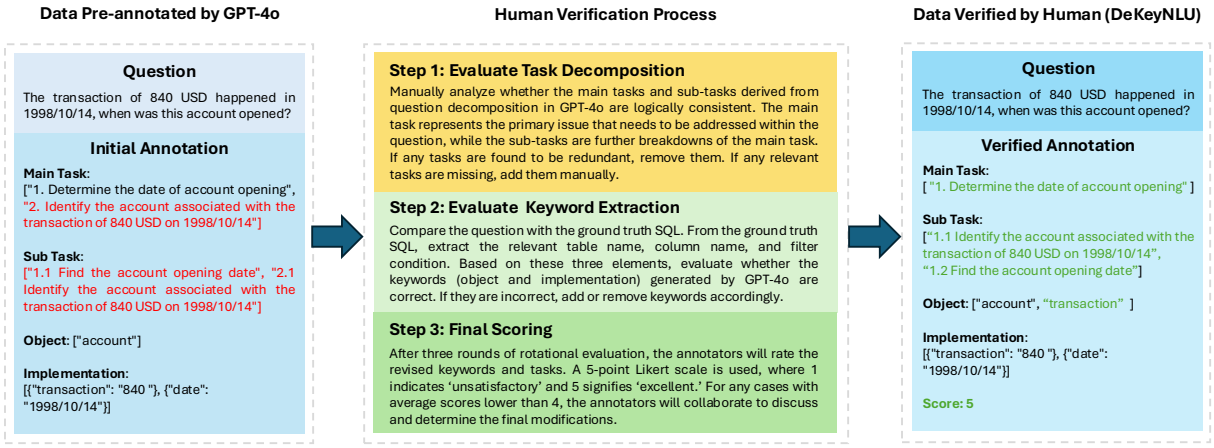


Figure 2: The DeKeyNLU dataset creation workflow. User questions are initially pre-annotated by GPT-4o for tasks (main and sub-tasks), objects, and implementations. These preliminary annotations are then subjected to a rigorous human verification process, where annotators correct and refine both task decomposition and keyword extraction. This involves three rounds of cross-validation. Following this, a final scoring phase identifies any low-scoring annotations, which are then collaboratively reviewed and further refined to produce the final, high-quality DeKeyNLU dataset.

(objects and implementations) were compared against user questions and ground truth SQL elements (filters, table/column names). Missing keywords were added, and extraneous ones removed. An initial training on 50 data points helped calibrate annotators and establish quality standards.

**3. Final Scoring:** After three rotational evaluation rounds, annotators rated revised keywords and tasks on a 5-point Likert scale (1=unsatisfactory, 5=excellent). Cases averaging below 4 were discussed collaboratively for final modifications. The inter-annotator agreement (Krippendorff's Alpha) for human verification was 0.762, indicating a high level of consistency.

### 3.3 Dataset Statistics

After three review rounds, a dataset of 1,500 question-answer pairs (decomposed tasks and keywords) was finalized. It was partitioned into training (70%), validation (20%), and testing (10%) sets for robust model development and evaluation. Figure 3 shows the distribution of main tasks, sub-tasks, and keywords, which indicate question complexity. A higher number of tasks tests reasoning and integration capabilities, while more keywords suggest intricate table/column setups prone to errors. For main tasks: 68.2% of questions have one task, 24.9% have two, and 6.9% have three or more. For sub-tasks: 31.7% comprise one to two sub-tasks, 60% have three to four, and 8.3% contain over five. For keywords: 20.3% are linked to one or two keywords, 60.6% to three or four, and 19.2% to five or more.

## 4 RAG-based NL2SQL Framework: DeKeySQL

We introduce DeKeySQL, a novel RAG-based framework for NL2SQL generation, designed to address common issues in existing approaches like MAC-SQL [Wang et al., 2023] and CHESS [Talaei et al., 2024], such as long runtimes, high costs, and accuracy limitations. As depicted in Figure 4, DeKeySQL comprises three main components: User Question Understanding (UQU), Entity Retrieval, and Generation.

### 4.1 User Question Understanding (UQU)

The initial phase of DeKeySQL focuses on comprehending user questions (Figure 4). The user question is incorporated into a prompt template and fed to an LLM (fine-tuned on DeKeyNLU) to generate a structured response encompassing two key tasks: Task Decomposition and Keyword Extraction.

**Task Decomposition:** Inspired by CoT reasoning [Wei et al., 2022], we decompose complex user questions into manageable components. We employ a two-level CoT approach, breaking questions into a *main task* (primary goal) and *sub-tasks* (refinements). This hierarchical structure aids the generation model by clarifying task dependencies. For example, the main task often corresponds to the main SELECT component in SQL, while sub-tasks map to operations like INNER JOIN, WHERE, etc. General LLMs like GPT-4o can falter here (Figure

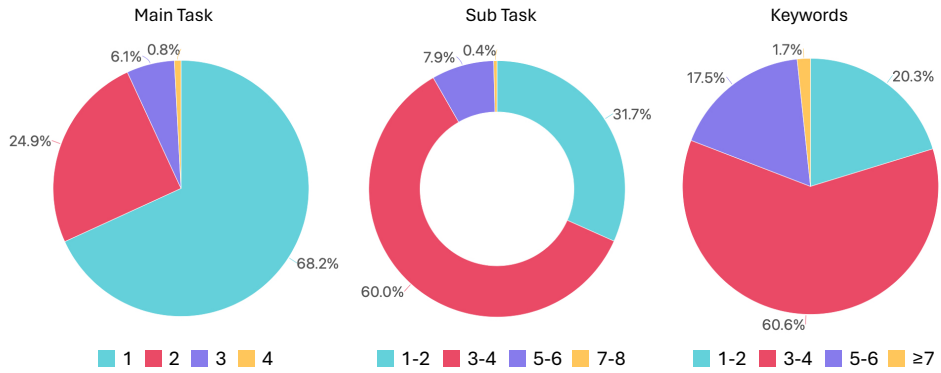


Figure 3: Distribution of the number of main tasks, sub-tasks, and keywords per question in the DeKeyNLU dataset. These distributions illustrate the complexity inherent in the questions, reflecting the reasoning and integration capabilities required of NL2SQL models.

2); thus, supervised fine-tuning with DeKeyNLU is employed to enhance stability and reliability.

**Keyword Extraction:** Unlike prior methods that simply broke sentences into keywords leading to irrelevancy, we classify keywords into *object* (terms associated with table/column names) and *implementation* (filtering criteria as a key-value dictionary). While In-Context Learning (ICL) with multiple examples can guide LLMs, models like GPT-4o may still generate irrelevant keywords (Figure 2). To mitigate this, we fine-tune smaller models like Mistral-7B [Jiang et al., 2023] using DeKeyNLU, enhancing keyword extraction accuracy.

### 4.2 Entity Retrieval

Following keyword extraction, this module retrieves corresponding database entities: table names, column names, table values, and textual descriptions (column/value descriptions). It consists of an embedder, retriever, and re-ranker. All table data are initially encoded and stored in a Chroma database. Keywords from UQU are encoded by the embedder and then used by the retriever to find the top-five resembling entities from the database. These are then passed to a re-ranker, which recalculates similarity scores and selects the two most similar entities. This process is divided into two sub-tasks:

**Database Retrieval:** Retrieves column names and table values. To handle large volumes of database values efficiently, we use MinHash [Zhu et al., 2016] + Jaccard Score or BM25 [Robertson et al., 2009]. MinHash generates fixed-size signatures for sets, approximating Jaccard similarity, while BM25 is a probabilistic model using term frequency and inverse document frequency. For column names, the top five scoring entities (score > 0) are selected. For purely numeric keywords, only exact matches for table values are considered; for mixed text/numeric keywords, the top five scoring entities are selected without a threshold. These are re-ranked to find the two most similar entities. Retrieved entities are cross-referenced to get table/column names, then de-duplicated and categorized (Figure 4).

**Textual Description Retrieval:** Retrieves column and value descriptions. Given a smaller dataset for this task, we directly use an embedding model to encode data, then cosine similarity in the retriever to find the top five entities. A specialized re-ranker model then determines the final relevance order.

### 4.3 Generation

This process has two phases: SQL Generation and Revision.

**SQL Generation:** Using ICL, general LLMs like GPT-4o [OpenAI, 2024b] generate initial SQL statements. Prompts (Appendix Figure 6) are structured into: data schema (formats, names, examples from Entity Retrieval), user question reasoning (question, main/sub-tasks from UQU, hints from dataset), constraints, and incentives. These details guide the model to produce an initial SQL statement.

**Revision:** Initial SQL may contain errors (incorrect table names, misaligned columns, etc., see Figure 4). Erroneous SQL and corresponding error messages are fed back to an LLM for revision (Appendix Figure 7). This iterative process yields syntactically correct, operational SQL queries.

## 5 Experiments

We conducted comprehensive experiments to evaluate the DeKeyNLU dataset and the DeKeySQL

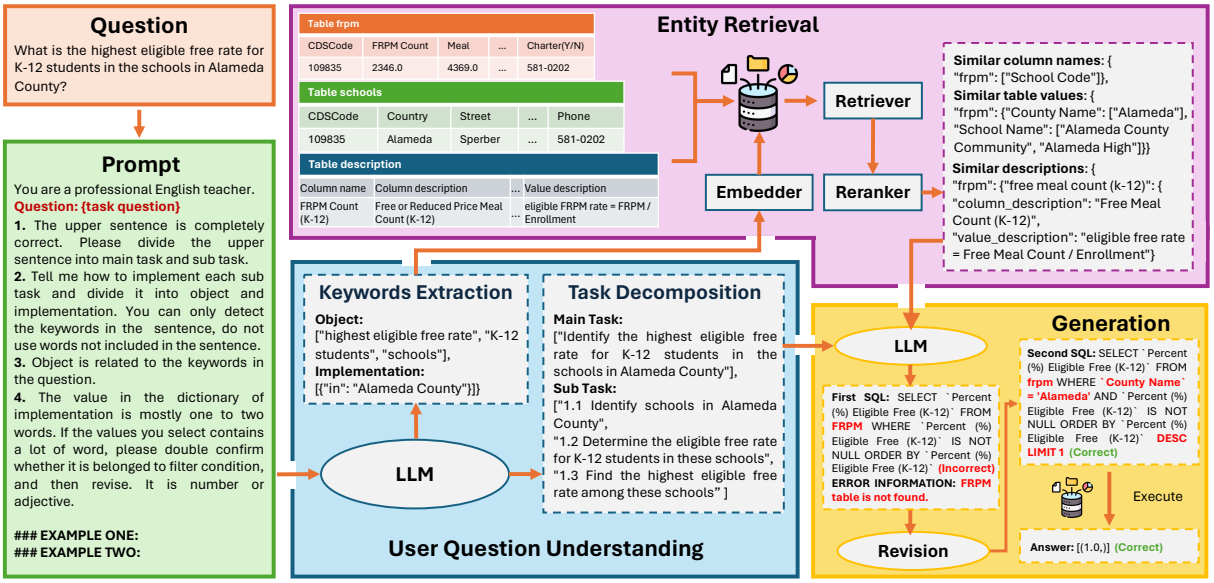


Figure 4: The DeKeySQL Framework. (1) The user's question is processed by the User Question Understanding (UQU) module using a prompt template, directing an LLM (fine-tuned on DeKeyNLU) to perform keyword extraction and task decomposition. (2) Extracted keywords are fed to the Entity Retrieval module to identify relevant column names, table values, and descriptions from the database. (3) Task decomposition outputs, retrieved entity data, and the original question are then input to the Generation LLM to produce SQL code. (4) If errors occur, the error information and generated SQL are passed to a revision LLM for correction. (5) Finally, the corrected SQL is executed to obtain the answer.

system. Our aims were: (1) to demonstrate DeKeyNLU's effectiveness for model fine-tuning, and (2) to assess DeKeySQL's performance against open-source SOTA methods.

### 5.1 Experiment Setting

**Datasets:** We used three datasets: DeKeyNLU (our proposed dataset), the BIRD development dataset, and the Spider dataset.

**NL2SQL Baseline Selection:** We selected open-source NL2SQL methods or those with published papers, including GPT-4o as a baseline. Methods include: Distillery [Maamari et al., 2024] (schema linking augmentation), CHESS [Talaei et al., 2024] (integrates data catalogs/values), MAC-SQL [Wang et al., 2023] (multi-agent framework), Dail-SQL [Gao et al., 2023] (prompt engineering with advanced question/example handling), and CodeS-15B [Li et al., 2024c] (incremental pre-training on SQL-centric corpus).

**Base Model Selection:** For UQU, models included GPT-4o-mini [OpenAI, 2024a], GPT-4 [Achiam et al., 2023], Mistral-7B [Jiang et al., 2023], LLaMA3-8B [Dubey et al., 2024], Baichuan2-7B, and 13B [Yang et al., 2023]. For entity retrieval, MinHash [Zhu et al., 2016] + Jaccard Score was compared against BM25 [Robertson et al., 2009]. Embedding models assessed: text-embedding-3-large [OpenAI, 2024c], Stella-1.5B, and Stella-400M. For code generation fine-tuning: DeepSeek-Coder-V2-Instruct, DeepSeek-Coder-V2-Base [Zhu et al., 2024], and Qwen-1.5-Coder [Yang et al., 2024].

| Method | Task Decomposition | | | Keyword Extraction |
|---|---|---|---|---|
| | BLEU | ROUGE | GPT-4o | F1 Score |
| Llama3-8B | 0.679 | 0.813 | 4.141 | 0.677 |
| Baichuan2-7B | 0.616 | 0.697 | 4.112 | 0.511 |
| Baichuan2-13B | 0.622 | 0.722 | 4.124 | 0.583 |
| Mistral-7B | 0.706 | 0.798 | 4.081 | **0.696** |
| GPT-4o-mini | 0.713 | 0.811 | 4.256 | 0.672 |
| GPT-4 | **0.722** | **0.816** | **4.286** | 0.665 |

Table 1: Performance comparison of various fine-tuned models on task decomposition (BLEU, ROUGE, GPT-4o score) and keyword extraction (F1 Score) using the DeKeyNLU test set. GPT-4 leads in task decomposition metrics, while Mistral-7B shows the best F1 score for keyword extraction.

**Fine-tuning Process:** Conducted on 4 Nvidia 4090 GPUs using Distributed Data Parallel and DeepSpeed. Uniform batch size of 1, epoch count of 1, learning rate of 2e-4. Low-Rank Adaptation (LoRA) [Hu et al., 2021] was used with rank=64, alpha=16, dropout=0.05. Bit precision was 4. Fine-tuning a UQU model with DeKeyNLU took 30 minutes; a code generation model took 4-5 hours.

| Method Configuration | Dev EX |
|---|---|
| UQU + Entity Retrieval + Revision + Generation | 60.36 |
| Entity Retrieval + Revision + Generation | 55.28 |
| Entity Retrieval + Generation | 51.25 |
| Generation Only | 46.35 |

Table 2: Module ablation study for DeKeySQL with GPT-4 as the backbone on the BIRD development set, showing Dev EX improvement with each added component. The full pipeline (UQU + Entity Retrieval + Revision + Generation) achieves the highest accuracy.

### 5.2 Metrics

**BLEU, ROUGE, and GPT-4o Score:** For evaluating task decomposition in NLU, we compared generated reasoning results against human-labeled ground truth using BLEU (BLEU-1, BLEU-2 for linguistic accuracy via n-gram matches) [Papineni et al., 2002], ROUGE (ROUGE-1, ROUGE-2, ROUGE-L for n-gram, sequence, and word pair overlap, measuring comprehensiveness/relevance) [Lin, 2004], and GPT-4o scores (five-point Likert scale, calibrated with human judgment for overall similarity) [Zheng et al., 2023]. Calibration details are in Table 9.

**F1 Score:** For keyword extraction in NLU, performance was evaluated using precision, recall, and the F1 score, balancing correctness and recall for a holistic view of extraction efficiency.

**Execution Accuracy (EX):** SQL query correctness was measured by comparing executed predicted query results against reference query results on specific database instances. This ensures semantic correctness and accounts for varied SQL formulations yielding identical results.

### 5.3 Results

**Supervised Fine-tuning with DeKeyNLU:** As shown in Table 4 (top row vs third row for UQU impact), fine-tuning the UQU module with DeKeyNLU elevated Dev EX from 62.31% (GPT-4o without DeKeyNLU fine-tuning for UQU) to 69.10% (GPT-4o-mini fine-tuned on DeKeyNLU for UQU, with GPT-4o for generation) on the BIRD dev dataset. On the Spider dev dataset, a similar improvement from 84.2% to 88.7% was observed. Table 1 reveals that model size impacts suitability for different fine-tuning tasks. Larger models (GPT-4, GPT-4o-mini) perform better on complex tasks like task decomposition after fine-tuning. Smaller models (Mistral-7B) outperform on tasks not requiring deep understanding, like keyword extraction. This suggests task-specific model selection for fine-tuning is crucial. Table 3 shows the effects of varying dataset sizes and epochs on keyword extraction fine-tuning. Mistral-7B performed best overall, followed by LLaMA-8B. For all models except Mistral-7B, F1-Score initially rose then fell with increasing training data, indicating more data isn"t always better. Increasing epochs consistently improved F1-Scores, suggesting it's a highly effective method for enhancing keyword extraction accuracy.

**Ablation Study:** The module ablation study (Table 2), using GPT-4 as the backbone, showed significant accuracy improvements with each added module. The UQU module (keyword extraction and task decomposition) yielded the largest gain, boosting accuracy by 9.18% (from 51.25% to 60.36%, comparing "Entity Retrieval + Generation" with the full pipeline). The entity retrieval module also contributed substantially, increasing accuracy by 4.9% (from 46.35% to 51.25%, comparing "Generation" with "Entity Retrieval + Generation"). UQU, entity retrieval, and revision modules were all indispensable. The model ablation study (Table 4) indicated MinHash outperformed BM25 in entity retrieval (e.g. 51.25% vs 49.34% when other components are GPT-4 and text-embedding-3-large) with less computation time. Surprisingly, the smaller Stella-400M embedding model surpassed the larger Stella-1.5B (e.g., 53.17% vs 51.36% Dev EX with GPT-4 UQU/Gen and MinHash retriever), suggesting parameter size isn"t always a guarantor of better performance. For code generation, general LLMs like GPT-4o and GPT-4 outperformed the fine-tuned smaller code models in our setup, underscoring the impact of parameter size and pre-training quality on complex code generation accuracy. These findings emphasize balancing architecture, parameter size, and task-specific optimization.

**BIRD and Spider Dataset Evaluation:** For BIRD, we report Dev EX due to its anonymity policy for test set evaluation; test EX and VES will be added in future updates. Table 5 shows DeKeySQL achieves the best Dev EX on BIRD compared to other SOTA models and is the current best open-source method. On Spider, DeKeySQL shows the highest EX on both dev and test sets. In practical utility assessment (Table 6), DeKeySQL excels in time efficiency, operational cost, and accuracy. Compared to CHESS, DeKeySQL achieves a 52.4% runtime reduction and a 97% operational cost decrease, showcasing significant industrial application potential. DeKeySQL is the top-

| Method | Dataset Size | | | | | Epoch | | | w/o Fine-tuning |
|---|---|---|---|---|---|---|---|---|---|
| | 20% | 40% | 60% | 80% | 100% | 1 | 2 | 3 | |
| Llama3-8B | 0.609 | 0.636 | **0.677** | 0.661 | 0.653 | 0.677 | 0.728 | 0.734 | 0.442 |
| Baichuan2-7B | 0.497 | 0.515 | 0.558 | 0.522 | 0.511 | 0.511 | 0.648 | 0.688 | 0.208 |
| Mistral-7B | **0.648** | **0.640** | 0.634 | **0.694** | **0.696** | **0.696** | **0.755** | **0.769** | **0.502** |
| Baichuan2-13B | 0.412 | 0.554 | 0.573 | 0.638 | 0.585 | 0.585 | 0.609 | 0.647 | 0.266 |

Table 3: Impact of DeKeyNLU dataset size (percentage of training data used) and number of training epochs on keyword extraction F1 score for various models. Mistral-7B generally benefits from more data and epochs, while other models show nuanced responses to dataset size.

| Module Focus | UQU Model | Entity Retrieval (Retriever + Embedder) | Generation Model | Dev EX |
|---|---|---|---|---|
| **UQU** | GPT-4o-mini (Finetuned on DeKeyNLU) | MinHASH + Stella-400M | GPT-4o | **69.10** |
| | Mistral-7B (Finetuned on DeKeyNLU) | MinHASH + Stella-400M | GPT-4o | 65.16 |
| | GPT-4o (No DeKeyNLU fine-tuning) | MinHASH + Stella-400M | GPT-4o | 62.31 |
| | GPT-4 (No DeKeyNLU fine-tuning) | MinHASH + Stella-400M | GPT-4o | 59.62 |
| **Generation** | GPT-4 | MinHASH + Stella-400M | GPT-4o | 59.62 |
| | GPT-4 | MinHASH + Stella-400M | DeepSeek-Coder-V2-Instruct (Finetuned) | 55.78 |
| | GPT-4 | MinHASH + Stella-400M | DeepSeek-Coder-V2-Base (Finetuned) | 50.41 |
| | GPT-4 | MinHASH + Stella-400M | Qwen-1.5-Coder (Finetuned) | 30.82 |
| | GPT-4 | MinHASH + Stella-400M | GPT-4 | 53.17 |
| **Entity Retrieval** | GPT-4 | MinHASH + Stella-400M | GPT-4 | 53.17 |
| | GPT-4 | MinHASH + Stella-1.5B | GPT-4 | 51.36 |
| | GPT-4 | MinHASH + text-embedding-3-large | GPT-4 | 51.25 |
| | GPT-4 | BM25 + text-embedding-3-large | GPT-4 | 49.34 |

Table 4: Performance (Dev EX on BIRD) of DeKeySQL with different backbone models for UQU, Entity Retrieval, and Generation modules. Results highlight the impact of DeKeyNLU fine-tuning (e.g., GPT-4o-mini for UQU) and the surprising efficacy of smaller embedding models like Stella-400M.

| Method | BIRD Dataset | Spider Dataset | |
|---|---|---|---|
| | Dev EX | Dev EX | Test EX |
| GPT-4 | 46.35 | 74.0 | 67.4 |
| Distillery [Maamari et al., 2024] | 67.21 | - | - |
| CHESS [Talaei et al., 2024] | 68.31 | 87.2 | - |
| Dail-SQL [Gao et al., 2023] | 54.76 | 84.4 | 86.6 |
| SFT CodeS-15B [Li et al., 2024c] | 58.47 | 84.9 | 79.4 |
| MAC-SQL [Wang et al., 2023] | 57.56 | 86.7 | 82.8 |
| **DeKeySQL(ours)** | **69.10** | **88.7** | **87.1** |

Table 5: Performance comparison (Execution Accuracy - EX) on BIRD and Spider datasets. DeKeySQL demonstrates state-of-the-art results among evaluated methods, particularly on the development sets. "-" indicates results not reported or not applicable. Results are sourced from official leaderboards where available.

| Method | Time(s) | Dev EX | Cost (USD) |
|---|---|---|---|
| CHESS [Talaei et al., 2024] | 119.38 | 0.5 | 11 |
| TA-SQL | 57.92 | 0.5 | 0.41 |
| SFT CodeS-15B [Li et al., 2024c] | **35** | 0.4 | - |
| MAC-SQL [Wang et al., 2023] | 133.55 | 0.7 | 0.38 |
| Chat2Query | 680.96 | 0.6 | - |
| **DeKeySQL (ours)** | 56.81 | **0.8** | **0.32** |

Table 6: Practical utility metrics (Time, Dev EX, Cost) for NL2SQL methods using GPT-4o as the base generation model. DeKeySQL demonstrates a strong balance of efficiency and accuracy. "-" indicates data not available.

performing open-source NL2SQL method evaluated.

## 6 Conclusion

This paper introduced DeKeyNLU, a novel dataset of 1,500 annotated question-SQL pairs, specifically designed to tackle critical challenges in task decomposition and keyword extraction for NL2SQL systems. Built upon the BIRD dataset, DeKeyNLU furnishes domain-specific annotations and establishes a high-quality benchmark for evaluating and improving LLM performance in this domain. Our comprehensive experiments demonstrate that fine-tuning with DeKeyNLU significantly enhances SQL generation accuracy, with performance reaching 69.10% on the BIRD development set (an increase from 62.31%) and 88.7% on the Spider development set (up from 84.2%). We further observed that larger models like GPT-4o-mini are particularly adept at task decomposition, whereas smaller, more agile models such as Mistral-7B excel in keyword extraction. Within the NL2SQL pipeline, entity retrieval was identified as the most critical component for overall accuracy, followed by user question understanding and the revision mechanisms. These findings underscore the profound value of dataset-centric approaches and meticulous pipeline design in advancing the capabilities of NL2SQL systems, paving the way for intuitive and accurate data interaction for users.

## Limitations

While DeKeyNLU and DeKeySQL demonstrate considerable advancements, several limitations and avenues for future research remain. The primary

constraint is the DeKeyNLU dataset size (1,500 samples), a consequence of resource limitations. While meticulously curated, this size may affect the robustness and generalizability of UQU models, particularly for highly diverse real-world scenarios. Expanding this dataset, possibly through semi-automated annotation techniques or exploring data augmentation strategies tailored for structured NLU tasks, is a key future direction. The restricted availability of high-quality annotated data, often compounded by copyright concerns for source data, poses an ongoing challenge for dataset expansion and community sharing that the field must address. Our benchmarking was also constrained by computational resources, preventing experimentation with the largest available LLMs (e.g., DeepSeek-V2-Coder-236B, Llama3.1-70B) or their integration with more advanced RAG modules. Such larger models and components could potentially yield further accuracy and robustness improvements. Future work should evaluate these cutting-edge models and diverse RAG configurations to establish more comprehensive benchmarks. Additionally, exploring the generalization of DeKeyNLU-trained models to completely unseen database schemas and question types, beyond the scope of BIRD, would be valuable. Investigating adaptive task decomposition strategies that can dynamically adjust granularity based on query complexity is another promising research avenue.

## References


Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Humaid Al Naqbi, Zied Bahroun, and Vian Ahmed. 2024. Enhancing work productivity through generative artificial intelligence: A comprehensive literature review. *Sustainability*, 16(3):1166.

James Allen. 1988. *Natural language understanding*. Benjamin-Cummings Publishing Co., Inc.

Tom B Brown. 2020. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Ursin Brunner and Kurt Stockinger. 2021. Valuenet: A natural language-to-sql system that learns from database information. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2177–2182. IEEE.

Zhiyu Chen, Wenhu Chen, Charese Smiley, Sameena Shah, Iana Borova, Dylan Langdon, Reema Moussa, Matt Beane, Ting-Hao Huang, Bryan Routledge, et al. 2021. Finqa: A dataset of numerical reasoning over financial data. *arXiv preprint arXiv:2109.00122*.

Jan Deriu, Katsiaryna Mlynchyk, Philippe Schläpfer, Alvaro Rodrigo, Dirk Von Grünigen, Nicolas Kaiser, Kurt Stockinger, Eneko Agirre, and Mark Cieliebak. 2020. A methodology for creating question answering corpora using inverse data annotation. *arXiv preprint arXiv:2004.07633*.

Jacob Devlin. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Ben Eyal, Amir Bachar, Ophir Haroche, Moran Mahabi, and Michael Elhadad. 2023. Semantic decomposition of question and sql for text-to-sql parsing. *arXiv preprint arXiv:2310.13575*.

Haishuo Fang, Xiaodan Zhu, and Iryna Gurevych. 2024. Dara: Decomposition-alignment-reasoning autonomous language agent for question answering over knowledge graphs. *arXiv preprint arXiv:2406.07080*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

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*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *arXiv preprint arXiv:2106.09685*.

Mohd Javaid, Abid Haleem, and Ravi Pratap Singh. 2023. A study on chatgpt for industry 4.0: Background, potentials, challenges, and eventualities. *Journal of Economy and Technology*, 1:127–143.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Nengzheng Jin, Joanna Siebert, Dongfang Li, and Qingcai Chen. 2022. A survey on table question answering: recent advances. In *China Conference on Knowledge Graph and Semantic Computing*, pages 174–186. Springer.

Maël Jullien, Marco Valentino, and André Freitas. 2024. Semeval-2024 task 2: Safe biomedical natural language inference for clinical trials. *arXiv preprint arXiv:2404.04963*.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2021. A deep dive into deep learning approaches for text-to-sql systems. In *Proceedings of the 2021 International Conference on Management of Data*, pages 2846–2851.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, and Nan Tang. 2024a. The dawn of natural language to sql: Are we fully ready? *arXiv preprint arXiv:2406.01265*.

Haitao Li, Junjie Chen, Qingyao Ai, Zhumin Chu, Yujia Zhou, Qian Dong, and Yiqun Liu. 2024b. Calibraeval: Calibrating prediction distribution to mitigate selection bias in llms-as-judges. *arXiv preprint arXiv:2410.15393*.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024c. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024d. 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.

Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*, pages 74–81.

Hanmeng Liu, Leyang Cui, Jian Liu, and Yue Zhang. 2021. Natural language inference in context-investigating contextual reasoning over long texts. In *Proceedings of the AAAI conference on artificial intelligence*, volume 35, pages 13388–13396.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024. A survey of nl2sql with large language models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*.

Guixiang Ma, Chun-Ta Lu, Lifang He, S Yu Philip, and Ann B Ragin. 2017. Multi-view graph embedding with hub detection for brain network analysis. In *2017 IEEE International Conference on Data Mining (ICDM)*, pages 967–972. IEEE.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*.

Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. 2019. Adversarial nli: A new benchmark for natural language understanding. *arXiv preprint arXiv:1910.14599*.

OpenAI. 2024a. Gpt-4o mini: advancing cost-efficient intelligence. *https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/*.

OpenAI. 2024b. Hello gpt-4o. *https://openai.com/index/hello-gpt-4o/*.

OpenAI. 2024c. New embedding models and api updates. *https://openai.com/index/new-embedding-models-and-api-updates/*.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*, pages 311–318.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*.

Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X Sean Wang. 2024. Purple: Making a large language model a better sql writer. *arXiv preprint arXiv:2403.20014*.

Justin Reppert, Ben Rachbach, Charlie George, Luke Stebbing, Jungwon Byun, Maggie Appleton, and Andreas Stuhlmüller. 2023. Iterated decomposition: Improving science q&a by supervising reasoning processes. *arXiv preprint arXiv:2301.01751*.

Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: Bm25 and beyond. *Foundations and Trends® in Information Retrieval*, 3(4):333–389.

Shreya Shankar, JD Zamfirescu-Pereira, Björn Hartmann, Aditya Parameswaran, and Ian Arawjo. 2024. Who validates the validators? aligning llm-assisted evaluation of llm outputs with human preferences. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, pages 1–14.

Chang-You Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain-of-thought style prompting for text-to-sql. *arXiv preprint arXiv:2305.14215*.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Tomer Wolfson, Mor Geva, Ankit Gupta, Matt Gardner, Yoav Goldberg, Daniel Deutch, and Jonathan Berant. 2020. Break it down: A question understanding benchmark. *Transactions of the Association for Computational Linguistics*, 8:183–198.

Aiyuan Yang, Bin Xiao, Bingning Wang, Borong Zhang, Ce Bian, Chao Yin, Chenxu Lv, Da Pan, Dian Wang, Dong Yan, et al. 2023. Baichuan 2: Open large-scale language models. *arXiv preprint arXiv:2309.10305*.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Fei Yu, Hongbo Zhang, Prayag Tiwari, and Benyou Wang. 2023. Natural language reasoning, a survey. *ACM Computing Surveys*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Yuheng Zha, Yichi Yang, Ruichen Li, and Zhiting Hu. 2024. Text alignment is an efficient unified model for massive nlp tasks. *Advances in Neural Information Processing Systems*, 36.

Xiaokang Zhang, Jing Zhang, Zeyao Ma, Yang Li, Bohan Zhang, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, Daniel Zhang-Li, et al. 2024. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *arXiv preprint arXiv:2403.19318*.

Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. Sciencebenchmark: A complex real-world benchmark for evaluating natural language to sql systems. *arXiv preprint arXiv:2306.04743*.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623.

Erkang Zhu, Fatemeh Nargesian, Ken Q Pu, and Renée J Miller. 2016. Lsh ensemble: Internet-scale domain search. *Proceedings of the VLDB Endowment*, 9(12).

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. 2021. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *arXiv preprint arXiv:2105.07624*.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*.

## A Disclaimers

DeKeyNLU is developed based on the BIRD dataset. Given the BIRD dataset's claim that it should be distributed under **CC BY-NC 4.0** [Li et al., 2024d].

The DeKeyNLU dataset will be publicly available under the same **CC BY-NC 4.0** license.

All annotators in our team hold bachelor's degrees with their education conducted in English. They were compensated at a rate of 14 USD/hr for their annotation work. The DeKeyNLU dataset does not contain any personally identifiable information or offensive content.

The DeKeyNLU dataset was initially generated by an LLM and then meticulously annotated and revised by human experts. After three rounds of manual revision, the high-quality DeKeyNLU dataset was finalized.

## B Performance of Revision Module

In our analysis of DeKeySQL, the Revision module is activated only once during processing. While it

enhances accuracy, multiple iterations do not necessarily lead to better outcomes proportionally to cost. We experimented on 50 sample queries, varying the revision threshold from 1 to 5 (Table 7). Increasing the threshold generally improves accuracy with an associated rise in computational cost, though execution time doesn"t follow a consistent pattern. A threshold of 3 offered an optimal balance of cost, accuracy, and execution time for our setup. For instance, a revision threshold of 1 yielded a Dev EX of 67.28% (on the BIRD dev set, based on context of DeKeySQL performance improvements being on BIRD), while increasing it to 5 improved Dev EX to 69.10%. The Revision module is capped at a threshold of 5 to prevent infinite loops and manage cost-effectiveness.

| Threshold | Time (s) | Cost (USD) | Accuracy (%) |
|---|---|---|---|
| One | 322.79 | 1.402 | 48 |
| Two | 357.57 | 1.598 | 58 |
| Three | 339.44 | 2.953 | 62 |
| Four | 345.23 | 3.119 | 62 |
| Five | 469.04 | 4.265 | 64 |

Table 7: Performance of the Revision module with different iteration thresholds on a sample of 50 queries from the BIRD dev set. Accuracy refers to Dev EX.

## C Error Analysis Details

Previous research, such as CHESS and CHASE-SQL, has not disclosed the specific datasets used for their error analyses, making direct objective comparisons challenging. Therefore, we conducted our own error analysis by randomly sampling 20% of failed cases from the BIRD dataset results. As shown in Table 8, we found that 45% of the golden SQL queries themselves had issues, primarily incorrect column names (11%) and missing GROUP BY/DISTINCT/RANK clauses (8%). Additionally, 6% of golden SQLs did not follow provided evidence cues. For DeKeySQL, 49% of its incorrectly generated SQL queries were mainly due to not adhering to evidence (17%), incorrect column usage (11%), and incorrect operations (8%). Vague questions, where question information was insufficient for correct SQL generation, accounted for 6% of issues, affecting both golden and predicted SQL.

### C.1 Error Types in Predicted SQL of DeKeySQL

Our error analysis (examples in Tables 10 through 14) identified five significant error types in DeKeySQL's predicted SQL:

| Error Category | % in Incorrect Golden SQL (Total 45%) | % in Incorrect Predicted SQL (Total 49%) | % in Vague Questions (Total 6%) |
|---|---|---|---|
| Evidence Misalignment | 6% | 17% | 0% |
| Incorrect Column | 11% | 11% | 5% |
| Incorrect Filtering | 5% | 4% | 1% |
| Description Issue | 0% | 0% | 0% |
| Incorrect Aggregation | 2% | 1% | 0% |
| Group by/Distinct/Rank Issue | 8% | 6% | 0% |
| Incorrect Operation | 6% | 8% | 0% |
| Date Handling Error | 0% | 0% | 0% |
| NULL Value Handling | 3% | 1% | 0% |
| Revision Error (Internal) | 0% | 0% | 0% |
| Incorrect Table | 4% | 1% | 0% |

Table 8: Distribution of error categories identified in Golden SQL queries, DeKeySQL's Predicted SQL queries (for failed cases), and Vague Questions from a 20% sample of BIRD dataset failed cases.

- **Incorrect Column Names:** DeKeySQL sometimes generates inaccurate column names or selects incorrect columns.
- **Incorrect Aggregation:** Occasionally, DeKeySQL joins tables unnecessarily or uses incorrect column names in the "ON" clause, leading to aggregation issues.
- **Incorrect Operation:** DeKeySQL may exhibit flawed or superfluous mathematical calculations, often due to an insufficient understanding of the database schema.
- **Incorrect Evidence Understanding:** In some instances, DeKeySQL fails to consult relevant evidence (e.g., formulas provided in prompts) when generating SQL commands, highlighting limitations in the LLM's adherence to complex instructions.
- **Incorrect Filtering:** Filtering values in SQL commands can be inaccurate or non-existent in the database. This is often linked to imprecise keyword extraction, indicating room for improvement in that sub-module.

### C.2 Errors in Keyword Extraction and Task Decomposition (Qualitative)

Qualitative examples of errors in keyword extraction and task decomposition from the DeKeyNLU annotation process are presented in Table 15 and 16. Keyword extraction errors are categorized as: missed keywords, wrong keywords, and useless keywords. Task decomposition errors include: a main task that should be a sub-task, an incomplete main task, an incomplete sub-task, incorrect sub-task numbering/assignment, and ambiguous sub-task definitions. These examples informed the refinement of our annotation guidelines and highlight the challenges LLMs face.

## D Calibration of GPT-4o Score for NLU Evaluation

To confirm the reliability of GPT-4o's automated evaluation for NLU tasks (task decomposition) and measure its alignment with human judgments, we incorporated calibration techniques. We compared GPT-4o scores for generated answers with human scores assigned by the three dataset annotators on a 5-point Likert scale. Table 9 summarizes results for six models. On average, human evaluations were consistently slightly higher (0.125 to 0.202, mean 0.152) than GPT-4o scores. To address this, inspired by methodologies from EvalGen [Shankar et al., 2024] and CalibraEval [Li et al., 2024b], we performed a calibration between GPT-4o scores and human evaluations. The resulting regression model is:

$$\text{HumanEvaluation} = 1.015 \times \text{GPT4oScore} + 0.042 \quad (1)$$

After applying this calibration (Equation 1), the average difference between calibrated GPT-4o scores and human evaluations reduced from 0.152 to 0.046, demonstrating significantly improved consistency.

| Model | GPT-4o Score | Human Evaluation | Difference (Human - GPT-4o) | Calibrated GPT-4o Score | Difference (Human - Calibrated) |
|---|---|---|---|---|---|
| Llama3-8B | 4.141 | 4.266 | 0.125 | 4.245 | 0.021 |
| Baichuan2-7B | 4.112 | 4.25 | 0.138 | 4.216 | 0.034 |
| Baichuan2-13B | 4.124 | 4.316 | 0.192 | 4.228 | 0.088 |
| Mistral-7B | 4.081 | 4.283 | 0.202 | 4.184 | 0.099 |
| GPT-4o-mini | 4.256 | 4.383 | 0.127 | 4.362 | 0.021 |
| GPT-4 | 4.286 | **4.416** | 0.13 | **4.392** | 0.024 |

Table 9: Comparison of Model Scores (GPT-4o automated score for task decomposition) and Human Evaluation on a 5-point Likert scale, before and after calibration. Calibration significantly reduces the difference between automated and human scores.

| | |
|---|---|
| **Question** | Please list the phone numbers of the direct charter-funded schools that are opened after 2000/1/1. |
| **Evidence** | Charter schools refers to Charter School (Y/N) = 1 in the frpm |
| **Gold SQL** | SELECT T2.Phone FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.**Charter Funding Type** = "Directly funded" AND T1.Charter School (Y/N) = 1 AND T2.OpenDate > "2000-01-01" |
| **Predicted SQL** | SELECT T1.Phone FROM schools T1 JOIN frpm T2 ON T1.CDSCode = T2.CDSCode WHERE T1.**FundingType** = "Directly funded" AND T1.OpenDate > "2000-01-01" AND T2.Charter School (Y/N) = 1 |

Table 10: Example of an incorrect column name error in DeKeySQL's predicted SQL. The model used "Funding-Type" from the 'schools" table alias T1, while the correct schema required "Charter Funding Type" from the "frpm" table alias T1 (or T2 if aliasing was different in gold).

| | |
|---|---|
| **Question** | Which school in Contra Costa has the highest number of test takers? |
| **Evidence** | "" |
| **Gold SQL** | SELECT sname FROM satscores WHERE cname = "Contra Costa" AND sname IS NOT NULL ORDER BY NumTstTakr DESC LIMIT 1 |
| **Predicted SQL** | SELECT T1.sname FROM satscores T1 **JOIN schools T2 ON T1.cds = T2.CDSCode** WHERE T2.County = "Contra Costa" AND T1.NumTstTakr IS NOT NULL ORDER BY T1.NumTstTakr DESC LIMIT 1 |

Table 11: Example of an incorrect aggregation error: DeKeySQL unnecessarily joined the 'schools" table when 'satscores" already contained sufficient information ("cname" for county, 'sname" for school name).

| | |
|---|---|
| **Question** | How many test takers are there at the school/s whose mailing city address is in Fresno? |
| **Evidence** | "" |
| **Gold SQL** | SELECT **T1.NumTstTakr** FROM satscores AS T1 INNER JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T2.MailCity = "Fresno" |
| **Predicted SQL** | SELECT **SUM(T2.NumTstTakr)** AS totalTestTakers FROM schools T1 JOIN satscores T2 ON T1.CDSCode = T2.cds WHERE T1.MailCity = "Fresno" |

Table 12: Example of an incorrect operation error: DeKeySQL used 'sUM(T2.NumTstTakr)" when the question asked for the number of test takers at *each* school, implying individual counts, not a sum across all schools matching the criteria.

| | |
|---|---|
| **Question** | Name schools in Riverside which the average of average math score for SAT is grater than 400, what is the funding type of these schools? |
| **Evidence** | **Average of average math = sum(average math scores) / count(schools).** |
| **Gold SQL** | SELECT T1.sname, T2.Charter Funding Type FROM satscores AS T1 INNER JOIN frpm AS T2 ON T1.cds = T2.CDSCode WHERE T2.District Name LIKE "Riverside%" GROUP BY T1.sname, T2.Charter Funding Type HAVING **CAST(SUM(T1.AvgScrMath) AS REAL) / COUNT(T1.cds) > 400** |
| **Gold SQL** | SELECT T1.sname, T2.FundingType FROM satscores AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T2.County = "Riverside" GROUP BY T1.sname, T2.FundingType HAVING **CAST(SUM(T1.AvgScrMath) AS REAL) / COUNT(T1.cds) > 400** |
| **Predicted SQL** | SELECT T1.sname, T2."FundingType" FROM satscores AS T1 JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T2.County = "Riverside" GROUP BY T1.sname, T2."FundingType" HAVING **AVG(T2.AvgScrMath) > 400** |

Table 13: Example of an incorrect evidence understanding error: DeKeySQL used "AVG(T2.AvgScrMath)" instead of following the provided evidence formula: "Average of average math = sum(average math scores) / count(schools)".

| | |
|---|---|
| **Question** | How many schools have an average SAT verbal score greater than 500 for students in grade 10? |
| **Evidence** | "" |
| **Gold SQL** | SELECT COUNT(DISTINCT T1.sname) FROM satscores AS T1 JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T1.AvgScrVerbal > 500 AND T2.GradeLevel = "High School" |
| **Predicted SQL** | SELECT COUNT(T1.cds) FROM satscores AS T1 JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T1.AvgScrVerbal > 500 AND T2.GradeLevel = "100" |

Table 14: Example of an incorrect filtering error: DeKeySQL incorrectly used a numeric value "100" for "GradeLevel" which should have been a text value like "High School" or "Middle School" based on the database schema, indicating a mismatch between extracted keywords and actual database values.

You are a professional English teacher.
**Question: {task question}**
**1.** The upper sentence is completely correct. Please divide the upper sentence into main task and sub task.
**2.** Tell me how to implement each sub task and divide it into object and implementation. You can only detect the keywords in the sentence, do not use words not included in the sentence.
**3.** Object is related to the keywords in the question.
**4.** The value in the dictionary of implementation is mostly one to two words. If the values you select contains a lot of word, please double confirm whether it is belonged to filter condition, and then revise. It is number or adjective.
**5.** Please only respond with a JSON object structured as follows, don't change the keys name.

**### EXAMPLE ONE:**
{
'question':"Name schools in Riverside which the average of average math score for SAT is grater than 400, what is the funding type of these schools?",
'main task':["1. Name schools in Riverside which the average of average math score for SAT is grater than 400", "2. what is the funding type of these schools?"],
'sub task':["1.1 find the name of schools in Riverside",
"1.2 get the average math score of these school",
"1.3 calculate the average score of average math score of eah school.",
"1.4 find the school which the average of average math score for SAT is grater than 400",
"2.1 the funding type of these schools"],
'object':['Name schools','funding type', 'average math score for SAT','schools'],
'implementation':[{'in':'Riverside'}, {'is grater than':'400'}]
}

**### EXAMPLE TWO:**
{
'question': "How many units of item no.9 were sold in store no.1 in total in January, 2012?",
' main task': ["Determine the total units sold of item no.9 in store no.1 in January, 2012"],
'sub task': ["1.1 Identify store no.1",
"1.2 Identify item no.9",
"1.3 Track sales in January, 2012",
"1.4 Calculate total units sold of item no.9"],
'object': ['units', 'item no', 'store no'],
'implementation': [{'store no.': '1'}, {'item no.': '9'}, {'in': 'January, 2012'}]
}

Figure 5: Prompt of keyword extraction and task decomposition.

You are a data science expert.
Below, you are presented with a database schema and a question.
Your task is to read the schema, understand the question, and generate a valid SQLite query to answer the question.
Before generating the final SQL query think step by step on how to write the query.

**### Database Schema**
**{DATABASE_SCHEMA}**

This schema offers an in-depth description of the database's architecture, detailing tables, columns, primary keys, foreign keys, and any pertinent information regarding relationships or constraints.
Pay attention!!! Special attention should be given to the examples listed beside each column of data schema, as they directly hint at which columns are relevant to our query.

**### Constraints**
1. For key phrases mentioned in the question, we have provided the most similar values within the columns denoted by "-- examples" in front of the corresponding column names. This is a crucial hint indicating the correct columns to use for your SQL query.
2. pay attention!!! avoid using different column for the same object with different filter values.
3. pay attention!!! Don't write a wrong column in the SQL code. Please check whether the column is belong to the table again in the SQL.

**### Question:**
**{QUESTION}**

**### Steps that you should follow:**
**{Main Task}**
**{Sub Task}**
**{Hint}**

The main task, sub task and evidence are correct, please base on them generate final sql query, please strictly follow the main task, sub task and evidence.
If there is an equation in the evidence, please strictly follow the equation!!!
The amount of item SELECT in sql query depends on the number of main tasks. if there is only one main task, you should only SELECT one item related to the main task in the sql query.

Please respond with a JSON object structured as follows:
{{"SQL": "Your SQL query is here."}}

Figure 6: The prompt template used for the SQL Generation module within DeKeySQL. This template structures the input to the LLM, including database schema, user question, decomposed tasks, and extracted entities, guiding the model to produce an initial SQL statement.

Objective: Your objective is to make sure a query follows the database admin instructions and use the correct conditions.

**Database Schema:**
**{DATABASE_SCHEMA}**

**### Constraints**
1. When you need to find the highest or lowest values based on a certain condition, using ORDER BY + LIMIT 1 is prefered over using MAX/MIN within sub queries.
2. If predicted query includes an ORDER BY clause to sort the results, you should only include the column(s) used for sorting in the SELECT clause if the question specifically ask for them. Otherwise, omit these columns from the SELECT.
3. Predicted query should return all of the information asked in the question without any missing or extra information.
4. For key phrases mentioned in the question, we have provided the most similar values within the columns denoted by "-- examples" in front of the corresponding column names. This is a crucial hint indicating the correct columns to use for your SQL query.
5. If you are joining multiple tables, make sure to use alias names for the tables and use the alias names to reference the columns in the query. Use T1, T2, T3, ... as alias names.

**### Question:**
**{QUESTION}**

**### ERROR INFORMATION**
**{Error Infomation}**

**### Steps that you should follow:**
**{Main Task}**
**{Sub Task}**
**{Hint}**

**### Predicted query:**
**{SQL}**

Pay attention to the ERROR INFORMATION, based on the error revise the SQL query.
Think about whether the predicted query used the hint and evidence already, if not, use the hint and evidence in the sql query generation.

Please respond with a JSON object structured as follows (if the sql query is correct, return the query as it is):
{{"revised_SQL": "Your revised SQL query is here."}}

Figure 7: The prompt template used for the Revision module in DeKeySQL. This template provides the LLM with the erroneous SQL query and associated error messages, facilitating a targeted correction process to refine the SQL into a syntactically correct and operational query.

| Error Type | Question | Predicted | Ground Truth |
|---|---|---|---|
| **Miss Keyword** | Write the title and all the keywords of the episode that was aired on 3/22/2009. | object["title", "keywords"] | object["title", "keywords", "episode"] |
| **Wrong Keyword** | Write down the need statement of Family History Project. | object['statement"] | object["need statement"] |
| **Useless Keyword** | List the tax code and inspection type of the business named "Rue Lepic". | object["tax code", "business", "inspection type", "name"] | object["tax code", "business", "inspection type"] |

Table 15: Qualitative Examples of Keyword Extraction Errors

| Error Type | Question | Predicted | Ground Truth |
|---|---|---|---|
| **Main Task Belongs to Sub Task** | What is the rental price per day of the most expensive children's film? | main task["1. Identify the most expensive children's film", "2. Find the rental price per day of that film"] | main task["1. Find the rental price per day of the most expensive children's film"] |
| **Main Task Is Incomplete** | Which nation has the lowest proportion of people who speak an African language? Please state the nation's full name. | main task["Identify the nation with the lowest proportion of speakers of African languages"] | main task["1. Identify the nation with the lowest proportion of people who speak an African language", "2. State the full name of this nation"] |
| **Sub Task Is Incomplete** | Please list the names of the male students that belong to the navy department. | main task["1. List the names of the male students that belong to the navy department"]<br>sub task["1.1 identify male students", "1.2 filter students belonging to the navy department"] | main task["1. List the names of the male students that belong to the navy department"]<br>sub task["1.1 find the male students", "1.2 filter students with navy department", "1.3 list the names of these male students"] |
| **Sub Task Number Is Wrong** | For the business with great experience existed in Sun Lakes city, provide the user ID who gave review on it and user followers. | main task["1. Identify the business with great experience in Sun Lakes city", "2. Provide the user ID who gave review on it and user followers"]<br>sub task["1.1 identify the business with great experience in Sun Lakes city", "1.2 find the user ID of the person who gave review on this business", "1.3 get the number of user followers for this user"] | main task["1. Identify the business with great experience in Sun Lakes city", "2. Provide the user ID who gave review on it and user followers"]<br>sub task["1.1 find the business with great experience in Sun Lakes city", "2.1 identify the user ID who gave review on this business", "2.2 find the followers of these users"] |
| **Sub Task Is Ambiguous** | Is SuperSport Park located at Centurion? | main task["1. Is SuperSport Park located at Centurion?"]<br>sub task["1.1 verify the location of SuperSport Park"] | main task["1. Is SuperSport Park located at Centurion?"]<br>sub task["1.1 find the location of SuperSport Park", "1.2 check if the location is at Centurion"] |

Table 16: Qualitative Examples of Task Decomposition Errors