PRISM: Navigating Cost–Accuracy Trade-offs for NL2SQL

GAURAV TARLOK KAKKAR, Georgia Institute of Technology, USA
YEOUNOH CHUNG, Google, USA
FATMA ÖZCAN, Google, USA
STEVE MUSSMANN, Georgia Institute of Technology, USA
JOY ARULRAJ, Georgia Institute of Technology, USA

Large language models (LLMs) have achieved strong performance on natural language to SQL (NL2SQL) tasks, but their practical effectiveness depends on tuning a complex pipeline of interacting components. Real-world deployments must navigate a critical trade-off between execution accuracy and monetary cost, a factor that has been largely overlooked by prior work focused primarily on maximizing accuracy. Navigating this trade-off is non-trivial: the ideal configuration of components (e.g., LLM, prompting strategy, schema linking) is not only interdependent but also highly sensitive to the target database schema. This creates a challenging, schema-aware configuration tuning problem that lacks a systematic solution.

We present PRISM, a framework that systematically identifies high-accuracy, cost-efficient NL2SQL configurations tailored to each schema. Adopting an optimize-then-deploy strategy, PRISM first uses cost-aware Bayesian Optimization in an offline phase to efficiently explore the configuration space and curate a pool of high-performing pipelines. In an online phase, it deploys these configurations either as a single, cost-effective candidate or as an ensemble to maximize accuracy. Experiments on the BIRD benchmark demonstrate that PRISM achieves 69.48% execution accuracy in the single-candidate setting, improving accuracy by 2.34% over the strongest baseline while reducing cost by 92%. In the ensemble setting, PRISM boosts accuracy further to 74.9%.

CCS Concepts: • **Information systems** → **Question answering**; **Structured Query Language**.

Additional Key Words and Phrases: NL2SQL; Large Language Models (LLM); Bayesian Optimization; Cost-Accuracy Trade-offs; Schema Linking; In-Context Learning.

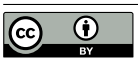
ACM Reference Format:
Gaurav Tarlok Kakkar, Yeounoh Chung, Fatma Özcan, Steve Mussmann, and Joy Arulraj. 2026. PRISM: Navigating Cost–Accuracy Trade-offs for NL2SQL. *Proc. ACM Manag. Data* 4, 1 (SIGMOD), Article 65 (February 2026), 27 pages. https://doi.org/10.1145/3786679

1 Introduction

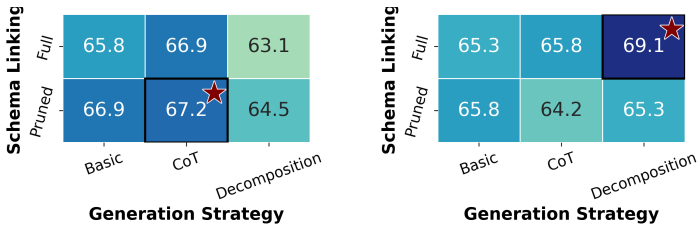
The ability to convert natural language (NL) questions into SQL has made data querying more accessible and powerful. NL2SQL techniques bridge the gap between technical database operations and non-expert users. Recent advances in large language models (LLMs) and in-context learning (ICL) have significantly improved the accuracy of NL2SQL systems [7, 11, 23, 26, 33, 34, 41]. However, these gains come with substantial monetary cost.

Modern NL2SQL systems rely on complex pipelines with numerous configurable choices, including the choice of LLM for generation, prompting strategies, schema linking methods, and the

Authors' Contact Information: Gaurav Tarlok Kakkar, Georgia Institute of Technology, USA, gkakkar7@gatech.edu; Yeounoh Chung, Google, USA, yeounoh@google.com; Fatma Özcan, Google, USA, fozcan@google.com; Steve Mussmann, Georgia Institute of Technology, USA, mussmann@gatech.edu; Joy Arulraj, Georgia Institute of Technology, USA, arulraj@gatech.edu.

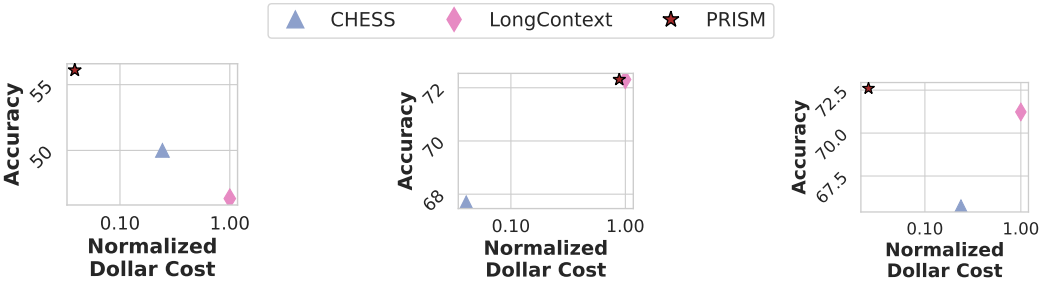


This work is licensed under a Creative Commons Attribution 4.0 International License.
© 2026 Copyright held by the owner/author(s).
ACM 2836-6573/2026/2-ART65
https://doi.org/10.1145/3786679



(a) Gemini-flash-2.0 **(b)** Gemini-pro-1.5

Fig. 1. Parameter interactions averaged across three databases from the BIRD benchmark. Heatmaps show execution accuracy (%) across combinations of schema linking and prompting strategies. Optimal choices vary significantly across models: (a) Gemini-flash-2.0 prefers a pruned schema with CoT prompting, while (b) Gemini-pro-1.5 achieves higher accuracy using the full schema and decomposition-based prompting.



(a) thrombosis_prediction (3 tables, 64 columns) **(b)** european_football_2 (7 tables, 199 columns) **(c)** toxicology (4 tables, 11 columns)

Fig. 2. Cost vs. accuracy analysis on three databases with varying schema complexity. Each plot compares LongContext, CHESS, and PRISM. Dollar cost is normalized with respect to LongContext for each database and shown on a logarithmic scale. PRISM consistently identifies configurations with better accuracy–cost trade-offs, highlighting the need for schema-specific optimization.

number of generated candidates. Each of these choices affects both the accuracy and the monetary cost. While prior research efforts have largely focused on improving the accuracy of the generated SQL queries, this paper treats monetary cost as a co-equal objective alongside accuracy in optimizing NL2SQL pipelines.

Finding a NL2SQL pipeline configuration that balances cost and accuracy is challenging: the space of possible configurations is large, and the optimal settings often depend on the database schema.

I - Parameter Interaction. The best choice for one pipeline component often depends on the values of others. Fig. 1 illustrates that the optimal choice of schema linking and prompting strategy is highly dependent on the LLM being used. The heatmaps show accuracy aggregated over three representative schemas from the BIRD benchmark [24] for two LLMs: Gemini-flash-2.0 and Gemini-pro-1.5. The optimal schema–prompt combination varies across models: Gemini-flash-2.0 prefers pruned schemas with prompting strategies like CoT (chain-of-thought reasoning) and basic, while Gemini-pro-1.5 performs best with full schemas and an advanced decomposition-based prompting (which breaks user questions into sub-problems). These interdependencies make manual tuning impractical.

II - Schema Dependency:. In addition to model-specific interactions, accuracy–cost trends also shift across database schemas, depending on their complexity and domain-specific requirements. For example, consider two state-of-the-art NL2SQL systems with fixed pipeline configurations: LongContext [7] and CHESS [41]. LongContext enriches prompt with synthetic examples and categorical values, leveraging LLMs with extended context windows (*e.g.*, Gemini). CHESS uses information retrieval and schema pruning, followed by decomposition-based prompting.

As shown in Fig. 2, neither system generalizes effectively: CHESS achieves higher accuracy at lower cost than LongContext on `thrombosis_prediction`, whereas LongContext outperforms CHESS on the other two schemas, but incurs higher cost. These inconsistencies reveal that fixed NL2SQL pipelines fail to account for database schema-specific characteristics (*e.g.*, number of tables and columns). Our analysis shows that significant performance gains are achievable through schema-specific tuning, highlighting the need for a systematic framework that can automatically adapt to each schema.

Our Approach. We introduce PRISM, a framework designed to systematically optimize NL2SQL pipelines by navigating trade-offs between execution accuracy and cost. We explicitly target schema-specialized tuning. PRISM formulates this as a *multi-objective configuration tuning problem* over a large combinatorial parameter space, focusing on the high-accuracy region of the Pareto frontier. To solve this problem efficiently, PRISM uses an optimize-then-deploy strategy:

Phase 1: In the offline phase, PRISM applies cost-aware Bayesian Optimization to explore the configuration space and identify a pool of high-performing pipelines. It then curates this pool based on the deployment mode: for single-candidate use, it builds a Pareto frontier of diverse, cost-efficient options; for ensemble use, it selects all configurations whose accuracy is within a tolerance of the highest-observed accuracy.

Phase 2: In the online phase, PRISM uses the selected configuration(s) to process incoming NL queries. In single-candidate mode, the chosen configuration generates the final SQL. In ensemble mode, each of the selected configurations generates a candidate, and a selection algorithm chooses the most reliable output.

Contributions.

- We introduce PRISM, a framework to perform systematic, schema-specific optimization of NL2SQL pipelines. Unlike prior work that proposes static, one-size-fits-all pipelines, PRISM automatically navigates the trade-offs between accuracy and cost to identify tailored configurations for individual schemas and user requirements.
- We propose an optimize-then-deploy framework that explores the NL2SQL configuration space via cost-aware Bayesian Optimization and curates configurations for two deployment modes: (i) a single-candidate mode that exposes a Pareto frontier of accuracy–cost trade-offs, and (ii) a multi-candidate ensemble mode that maximizes execution accuracy.
- PRISM outperforms strong baselines on the BIRD benchmark and surfaces diverse, schema-specific optimal settings. In the single candidate setting, PRISM achieves 69.48% execution accuracy, outperforming the strongest baseline (LongContext) by 2.34% while reducing the dollar cost by 92%. In the ensemble setting, where accuracy is prioritized over cost, it achieves 74.9% execution accuracy.

2 Background

2.1 Typical NL2SQL Pipeline

NL2SQL aims to translate natural language questions into SQL queries. Recent advancements in large language models (LLMs) have enabled the development of in-context learning (ICL) approaches [7, 10, 11, 23, 34, 35, 37, 41]. These techniques leverage schema information and additional

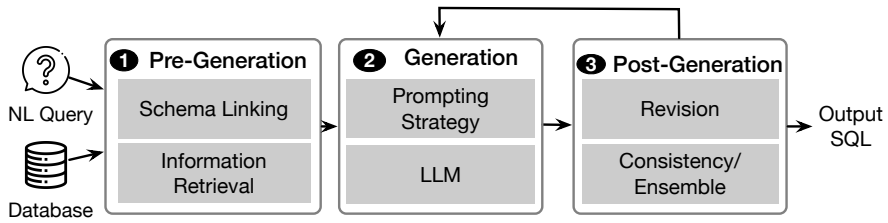


Fig. 3. Overview of a typical NL2SQL pipeline, illustrating the Pre-Generation, Generation, and Post-Generation stages.

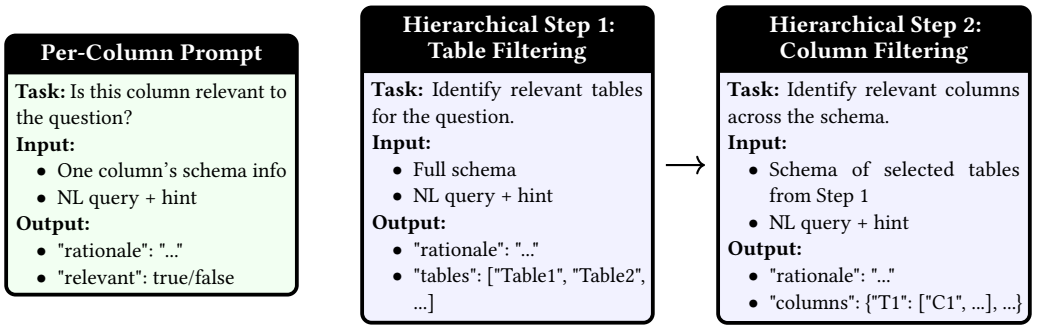


Fig. 4. Prompts used for schema linking. **Left:** Per-Column evaluates each column independently. **Right:** Hierarchical performs a two-stage process: first selecting relevant tables, then identifying relevant columns within those tables.

contextual data within the prompt to help LLMs generate accurate SQL queries. Fig. 3 depicts a typical NL2SQL pipeline, organized into three primary stages: Pre-Generation, Generation, and Post-Generation.

Pre-Generation. This stage gathers all relevant context required to answer the user query. Its key components include schema linking and information retrieval. Schema linking identifies relevant schema elements, such as tables and columns, needed to address the user query. The information retrieval component extracts additional details, such as database content retrieval, which identifies relevant column values for categorical attributes. Other contextual elements, such as demonstrative examples [7, 33] and domain-specific information [1, 33], further aid in NL2SQL translation.

Generation. In this stage, the LLM is prompted using the contextual information gathered in the previous stage and the chosen prompting strategy to generate the SQL query. Existing works have explored various prompting strategies, including CoT (Chain of Thought) [40, 48] and decomposition [27] approaches.

Post-Generation. This stage focuses on detecting and repairing potential errors in the generated SQL query. Several strategies have been proposed, including syntactic correction by prompting the language model with the error and requesting a correction [26]. Some approaches execute the generated query to obtain outputs and leverage execution-guided techniques to identify and address errors. For instance, NULL outputs can signal potential issues in the SQL query [7]. Another critical component of the post-generation stage is self-consistency and selection strategy. Here, multiple candidate SQL queries are generated for the same user query, followed by a voting [47] or selection [11, 33] mechanism to select the final query. This approach recognizes that complex

reasoning tasks may have multiple valid paths to the correct answer. By exploring diverse reasoning paths and selecting the most consistent result, it improves the quality of the generated SQL.

2.2 NL2SQL Key Parameters

1. **Prompting Strategy:** Strategies like CoT and decomposition influence both the quality and efficiency of SQL generation. In our experiments, we evaluate three prompting strategies: basic, CoT, and decomposition.
2. **Schema Linking Strategy:** Identifies relevant schema elements for the question. Besides using the full schema (FullSchema), strategies exist to prune schema [26, 29, 33, 41].
   - *LLM-based schema linking* uses LLM to assess relevance, either via the Per-Column strategy, which evaluates each column independently, or the Hierarchical strategy, which performs a two-stage table-then-column selection. The prompts used for each strategy are illustrated in Fig. 4. In Fig. 1, "pruned schema" setting corresponds to Hierarchical strategy.
   - *Hybrid linking* extends the LLM-based approach by incorporating keyword-based textual similarity methods (e.g., using keywords, substrings, and fuzzy matching). This results in two hybrid strategies: Hybrid Per-Column and Hybrid Hierarchical. The choice of strategy impacts the trade-off between accuracy (retrieving necessary schema elements vs. including noise) and efficiency (prompt length and LLM cost).

   In total, our work considers these five distinct schema linking strategies.
3. **Many-Shot Examples:** Varying the quantity and selection method (static vs. online) of many-shot query-SQL example pairs in the prompt affects both accuracy and cost. We experiment with 0 to 100 examples per query (*e.g.*, 10, 35, 50, 75, 100).
4. **LLM Models:** Different LLMs (e.g., Gemini-pro-1.5, Gemini-flash-2.0, Qwen-7B, Qwen-14B) vary in their reasoning capabilities, accuracy, and computational cost.
5. **Self-Consistency/Ensemble:** Using self-consistency or ensemble by generating multiple candidate SQL queries and selecting the most appropriate one improves robustness but increases latency and cost.
6. **Retry:** Employing retry mechanisms to re-prompt the model when errors are detected versus using a single-shot approach affects both accuracy and computational efficiency. We allow up to 3 retries in our experiments.
7. **Categorical Column Values:** Similar to prior work [41], we extract keywords from the question and use locality-sensitive hashing (LSH) [8] to retrieve similar column values from the database. These values are included in the schema to help the LLM generate more accurate SQL queries. As this consistently improves accuracy with negligible token overhead, we apply it across all pipelines.

These parameters span all major stages of the NL2SQL pipeline, pre-generation (*e.g.*, schema linking), generation (*e.g.*, prompting, LLM choice, many-shot examples), and post-generation (*e.g.*, retry, self-consistency), ensuring that our configuration space is comprehensive and captures the key design dimensions influencing both accuracy and cost.

Evaluation Metrics. To assess the effectiveness of the NL2SQL pipeline, the following metrics are considered:

1. **Execution Accuracy:** The correctness of the generated SQL query, measured by its ability to return the expected output when executed.

Table 1. Accuracy of models across schemas with other parameters fixed.

Database Schema	Model	Accuracy (%)
superhero	Gemini-pro-1.5	89.92
superhero	Gemini-flash-2.0	86.82
thrombosis_prediction	Gemini-pro-1.5	47.85
thrombosis_prediction	Gemini-flash-2.0	59.51

Table 2. Accuracy comparison across database schemas using the Hierarchical and FullSchema linking strategies for Gemini-pro-1.5, with other parameters held fixed.

Database Schema	Hierarchical	FullSchema
european_football_2	68.22	72.09
codebase_community	68.28	62.90

2. **Cost:** The computational cost associated with input and output tokens, particularly relevant for LLMs accessed via APIs.

The NL2SQL pipeline involves several interacting decisions across components and parameters. PRISM tackles the challenge of optimizing the pipeline configuration for both accuracy and cost.

3 Challenges

3.1 Schema-Specific Configuration Needs

Optimal configurations for NL2SQL pipelines vary significantly across database schemas due to differences in complexity and the necessity of domain-specific information. Databases with simpler schemas and well-described columns often perform well with basic generation strategies, while more complex schemas demand advanced strategies and detailed domain knowledge.

For example, in the bird benchmark [24], simpler databases such as superhero exhibit straightforward schemas with descriptive column names. These characteristics enable high accuracy with minimal context. In contrast, complex databases such as thrombosis_prediction pose significant challenges due to their reliance on domain knowledge. For instance, column names like "LDH" refer to lactate dehydrogenase, and normal ranges for uric acid vary by gender *e.g.*, (Male) and (Female). LLMs must interpret these domain-specific details to accurately answer user queries, making thrombosis_prediction substantially more difficult.

Providing domain-specific hints (available as part of the bird benchmark) also significantly impacts accuracy. For superhero, accuracy improves from 79% to 87% with hints. For thrombosis_prediction, the improvement is even more pronounced, rising from 20% to nearly 50%. These results underscore the critical role of domain knowledge in achieving high accuracy for complex databases.

Beyond domain knowledge, the choice of other core components must also be adapted per-database. As Tab. 1 highlights, the optimal LLM varies; while Gemini-pro-1.5 excels on superhero, the smaller Gemini-flash-2.0 model surprisingly performs better on the difficult thrombosis_prediction database. Similarly, schema linking strategies must be tailored. As shown in Tab. 2, providing the FullSchema is best for european_football_2, whereas an LLM-based filtering strategy (Hierarchical) is superior for codebase_community.

In summary, these schema-specific variances highlight the difficulty of developing a one-size-fits-all configuration and emphasize the need for systematic approaches tailored to the unique requirements of each schema.

3.2 Multi-Objective Trade-Offs

Balancing execution accuracy and cost is another significant challenge in optimizing NL2SQL pipelines. These objectives conflict, creating trade-offs that require careful consideration based on application requirements. Strategies that improve accuracy typically increase API token usage and, consequently, monetary cost.

Prompting strategies such as CoT and decomposition are more verbose than basic, increasing generation token usage and overall cost. On the superhero database, CoT produces responses that require 1.1× more tokens, while decomposition increases token usage by 3.0× compared to basic. These effects grow when column values or many-shot examples are included, highlighting how prompt design directly influences NL2SQL system's monetary cost.

Thus, the number of combinations that a developer must explore in this multi-objective trade-off space is large. We argue that there is a need for a systematic approach to navigate these trade-offs. Such an approach should efficiently explore the configuration space, identifying configurations that balance these competing objectives and align with application priorities.

4 PRISM Overview

4.1 Problem Formulation

The challenges outlined in § 3 underscore the difficulty of selecting optimal NL2SQL configurations across diverse databases and usage scenarios. The configuration space is large, evaluation is expensive, and performance objectives often conflict. To address this, we formulate the problem as a multi-objective optimization task over the space of pipeline configurations.

As discussed in § 2.1, a typical NL2SQL pipeline consists of three key stages: PRE-GENERATION, GENERATION, and POST-GENERATION. Each stage has a set of key configuration parameters controlling its behavior. Let k_i^s denote a parameter in stage $s \in \{\text{pre}, \text{gen}, \text{post}\}$, and let V_i^s be its finite domain. A full configuration $\theta \in \Theta$ is defined as:

$$\theta = (k_1^{\text{pre}}, \ldots, k_a^{\text{pre}}, k_1^{\text{gen}}, \ldots, k_b^{\text{gen}}, k_1^{\text{post}}, \ldots, k_c^{\text{post}})$$

where Θ is the Cartesian product of all parameter domains.

Given a database D, a labeled query dataset $\mathbb{Q} = \{(q_i, sql_i)\}_{i=1}^N$, an accuracy function $A(\theta, D, \mathbb{Q})$ measuring the percentage of correctly answered queries, and a computational cost function $C(\theta, D, \mathbb{Q})$ capturing monetary cost, the set of Pareto-optimal configurations Θ^* is defined as:

$$\Theta^* = \{\theta \in \Theta \mid \nexists \theta' \in \Theta : \theta' \succ \theta\}$$

where $\theta' \succ \theta$ denotes that θ' dominates θ, defined as:

$$\begin{aligned} \theta' \succ \theta \iff & [A(\theta') > A(\theta) \wedge C(\theta') \leq C(\theta)] \vee \\ & [A(\theta') \geq A(\theta) \wedge C(\theta') < C(\theta)] \end{aligned}$$

While the Pareto frontier Θ^* includes all non-dominated configurations, our focus lies specifically on the high-accuracy region. In practice, users are unlikely to adopt configurations that are cheap but yield unacceptably low accuracy. Therefore, our ultimate goal is to find a set of high-accuracy configurations that expose meaningful accuracy-cost trade-offs, rather than enumerating the entire frontier. These selected configurations serve as practical deployment options under varying budget constraints.

This formalization defines the solution space we aim to explore. In the remainder of this section, we describe how we approach this optimization using a two-phase design tailored to NL2SQL pipelines.

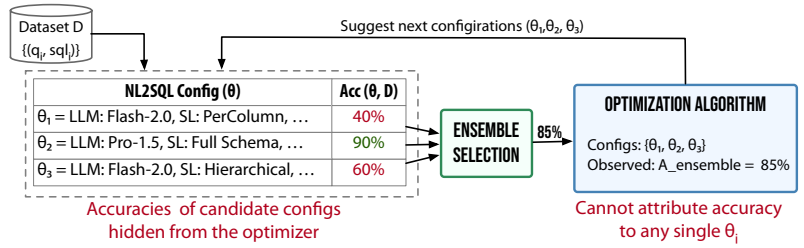


Fig. 5. Weak feedback in ensemble optimization. Although each candidate configuration θ_i yields a different accuracy (e.g., 40%, 90%, 60%), the optimizer observes only the ensemble's aggregate performance (85%). This weak feedback hinders its ability to identify and reward strong configurations, complicating optimization.

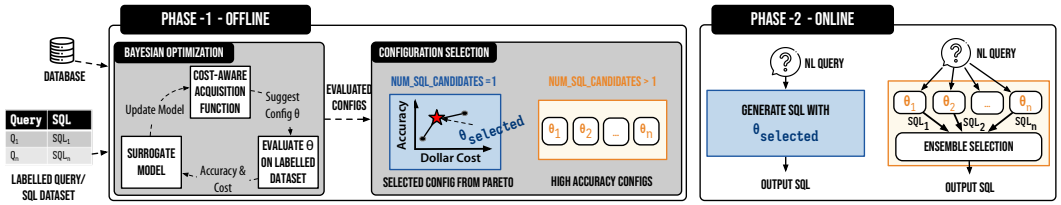


Fig. 6. Overview of PRISM's two-phase framework. Phase 1 (offline) consists of two stages: first, cost-aware Bayesian Optimization generates a pool of high-performing configurations. Second, a Configuration Selection step curates this pool for the target deployment mode. (`num_sql_candidates` = 1): A diverse set of high-accuracy configurations is distilled into a Pareto frontier, from which a single configuration ($\theta_{selected}$) is selected based on user requirements for deployment. (`num_sql_candidates` > 1): All configurations within an accuracy tolerance ϵ of the best are selected to form the ensemble. Phase 2 (online) then uses these pre-selected configurations to process incoming NL queries.

4.2 Two-Phase Optimization Framework

Optimizing NL2SQL pipelines becomes significantly more complex in the presence of multi-candidate ensembles. Many modern systems generate multiple candidate SQL queries (`num_sql_candidates` >1) per natural language question and consolidate them using techniques such as majority voting or fine-tuned selectors [11, 33]. However, these systems typically vary only a single dimension across candidates, such as the generation strategy (*e.g.*, basic, CoT, or decomposition) or the generation model. In contrast, an expressive configuration space should ideally allow each candidate to be generated from a potentially distinct pipeline configuration.

While the underlying configuration space is already large, increasing `num_sql_candidates` magnifies the complexity substantially. Instead of selecting a single configuration, the optimizer must now reason about interactions among multiple heterogeneous pipelines, each with different cost–accuracy trade-offs, yet contributing jointly to a single aggregated output. As illustrated in Fig. 5, suppose `num_sql_candidates` is set to 3, and each candidate uses a different configuration (*e.g.*, varying in LLM model, schema linking, *e.t.c.*). Each candidate configuration θ_i is evaluated on a labeled query set (D, Q) to compute its accuracy $A(\theta_i, D)$, representing the percentage of queries it answers correctly when executed. Even if only one configuration is strong, the ensemble may still perform well—causing the optimization algorithm to assign a high score to this trio, despite the others being suboptimal. Importantly, the optimizer does not observe the individual accuracies $A(\theta_i, D)$; it sees only the ensemble's aggregate execution accuracy, which acts as a shared reward signal for the group. This introduces two key challenges:

- **Weak Feedback:** Ensemble accuracy reflects the aggregated performance of multiple candidate configurations, making it difficult to attribute success or failure to any individual θ_i. This diluted feedback hinders credit assignment and slows optimizer convergence.
- **High Evaluation Cost:** Evaluating a single ensemble configuration requires multiple LLM invocations. As `num_sql_candidates` increases, the cost of each trial grows proportionally, significantly increasing the overall optimization expense.

To address these challenges, PRISM adopts the two-phase framework shown in Fig. 6. In **Phase 1**, we focus on identifying a diverse set of high-performing single-candidate pipelines by exploring the configuration space using cost-aware Bayesian Optimization. Then, in **Phase 2**, we leverage these strong candidates to support different deployment modes: either selecting a single configuration based on user constraints, or combining multiple configurations into an ensemble to boost accuracy and robustness.

Phase 1: Offline Exploration and Selection. As shown in the left half of Fig. 6, this phase takes a database schema and a labeled query–SQL dataset as input and performs two key steps:

- **Bayesian Optimization:** We apply cost-aware Bayesian Optimization to efficiently explore the configuration space. The surrogate model iteratively proposes configurations, which are evaluated on the labeled dataset for accuracy and cost, yielding a pool of strong candidates.
- **Configuration Selection:** The resulting pool is curated based on deployment mode: *Single-candidate* (`num_sql_candidates` = 1): high-accuracy configurations are filtered and clustered for diversity, yielding a Pareto frontier of cost–accuracy trade-offs from which users select θ_{selected}. *Ensemble* (`num_sql_candidates` > 1): We select all configurations from the pool whose accuracy is within a predefined tolerance, ε, of the highest-observed accuracy.

Phase 2: Online Deployment. This phase uses the configuration(s) curated in Phase 1 to process incoming NL queries. The deployment mode determines runtime behavior:

- **Single-Candidate Mode**: The selected configuration (θ_{selected}) is used to directly generate the final SQL query. This provides a low-cost, efficient solution.
- **Ensemble Mode:** Each configuration from the candidate pool generates a SQL query. A downstream selection strategy, described in § 5.2, then compares these candidates and selects the most likely correct output. This boosts accuracy and robustness in settings where additional runtime cost is acceptable.

Usage Scenario. The user provides PRISM with a small labeled dataset of representative question–SQL pairs for their specific database. Notably, the user does not provide any accuracy or cost constraints to the optimizer upfront. Instead, using the labeled data, PRISM completes its offline Phase 1 and presents the user with the discovered Pareto frontier, a menu of optimal configurations. For example:

- *Config A (High Accuracy):* 90% accuracy, $0.50/query
- *Config B (Balanced):* 88% accuracy, $0.10/query
- *Config C (Low Cost):* 85% accuracy, $0.02/query

The user then selects a configuration for online deployment based on their application's specific needs and budget. This approach ensures the final selection is made from a set of proven, cost–accuracy efficient configurations.

Ultimately, this two-phase framework enables PRISM to surface high-quality configurations through offline exploration and deploy them flexibly-either as a single efficient pipeline or as an ensemble for maximum accuracy.

Algorithm 1 Select Diverse High-Accuracy Configurations

Require: Evaluated configurations D_{eval} (params θ, accuracy $A(\theta)$), target diversity k, accuracy threshold ϵ
Output: Set of up to k diverse configurations D_{selected}

$A_{\max} \leftarrow \max_{\theta \in D_{\text{eval}}} A(\theta)$
$D_{\text{filtered}} \leftarrow \{\theta \in D_{\text{eval}} \mid A(\theta) \geq A_{\max} - \epsilon\}$ ▷ Filter high-accuracy candidates
if $|D_{\text{filtered}}| \leq k$ **then**
    return D_{filtered}
end if
$clusters \leftarrow \text{ClusterParams}(D_{\text{filtered}}, \text{k} = k)$ ▷ Cluster based on config params
$D_{\text{selected}} \leftarrow \text{SelectBestPerCluster}(D_{\text{filtered}}, clusters, \text{metric} = A(\theta))$ ▷ Select highest accuracy from each cluster
return D_{selected}

5 Algorithms

5.1 Phase 1: Configuration Exploration and Selection

Phase 1 consists of two main steps: (1) exploring the configuration space using Bayesian Optimization, and (2) selecting configurations from the resulting pool based on the intended deployment mode.

(1) Configuration Space Exploration via Bayesian Optimization. We use Bayesian Optimization to efficiently explore the configuration space, modeling the accuracy–cost surface with a Gaussian Process (GP) surrogate [49]. At each step, an acquisition function selects the next configuration to evaluate by balancing exploration and exploitation [18]. The goal of this exploration is to generate high-accuracy configurations across a range of cost regimes. To support this, we compare four acquisition functions:

Accuracy-Focused Expected Improvement (EI) [30]. Maximizes the expected gain in accuracy over the current best. While effective for locating high-performing regions, EI tends to favor expensive configurations and ignores cost.

Multi-Objective (Pareto-based) [32]. Treats accuracy and cost as separate objectives and seeks to recover the full Pareto frontier via hypervolume-based improvement. However, many evaluations are spent on low-cost, low-accuracy configurations that lie outside the preferred accuracy region.

Cost-Aware Acquisition Functions

- **EICool [21]:** Modifies EI by dividing expected improvement by configuration cost, with a cooling schedule that gradually reduces cost sensitivity. This allows an early focus on cost-effective configurations and later prioritization of accuracy.
- **Pareto-efficient Acquisition Function (CEI) [13]:** CEI restricts its focus to the top $\lambda\%$ of configurations based on expected improvement and selects the one with the lowest cost. We set $\lambda = 5$. This heuristic promotes cost-efficient selection and has demonstrated strong empirical performance.

In our experiments (§ 6.7), CEI consistently achieves the best balance across databases. Unlike EI, which can over-focus on a single promising region, CEI filters for high-EI candidates and then selects the cheapest, helping it avoid local optima. EICool starts with low-cost configurations but often gets stuck there, missing better-performing options at higher costs.

(2) Configuration Selection. Once the BO exploration concludes, we curate the pool of evaluated configurations based on the deployment mode:

Single-Candidate Deployment Many of the highest-accuracy configurations may be minor variants of each other, *e.g.*, differing only in the number of many-shot examples used (75 vs. 100) while keeping all other parameters the same. Such configurations offer limited value to users seeking meaningful trade-offs. Therefore, instead of selecting the top-k configurations by accuracy alone, we apply a clustering-based procedure (Alg. 1). We first filter for high-accuracy candidates

Algorithm 2 Ensemble Selection via Grouped Pairwise Scoring

Require: Candidate SQLs $\mathcal{C} = \{c_1, c_2, \ldots, c_n\}$, user question Q_u, hint H_u, target database D, selection model θ_p, $\texttt{er}(c_i, D)$ as the execution result of c_i on D
Output: Most likely correct candidate c^*

$\mathcal{C} \leftarrow \texttt{DeduplicateSQL}(\mathcal{C})$ ▷ Remove identical SQL candidates
Groups $\leftarrow \texttt{GroupBy}(\texttt{er}(c_1, D), \ldots, \texttt{er}(c_n, D))$
GroupScores$[g] \leftarrow 0$ for each group $g \in$ Groups
for each unordered pair (i, j) where $i < j$ **do**
  if $\texttt{er}(c_i, D) = \texttt{er}(c_j, D)$ **then**
    continue ▷ Skip identical outputs
  end if
  $S_{i,j} \leftarrow \texttt{SchemaUnion}(c_i, c_j, D)$ ▷ Schema elements referenced by both candidates
  $w_{ij} \leftarrow \theta_p(Q_u, H_u, S_{i,j}, c_i, \texttt{er}(c_i, D), c_j, \texttt{er}(c_j, D))$
  $w_{ji} \leftarrow \theta_p(Q_u, H_u, S_{i,j}, c_j, \texttt{er}(c_j, D), c_i, \texttt{er}(c_i, D))$
  if $w_{ij} = w_{ji}$ **then** ▷ Consistent winner
    $g \leftarrow \texttt{ResultToGroup}(\texttt{er}(c_{w_{ij}}, D))$
    GroupScores$[g] \leftarrow$ GroupScores$[g] + 1$
  end if
end for
$G^* \leftarrow$ group with highest score (break ties by group size, then random)
$c^* \leftarrow$ first candidate in G^*
return c^*

within an ϵ margin of the best observed accuracy, then cluster them based on their parameter values using the K-Prototypes algorithm [16], which supports mixed categorical and numerical features. From each cluster, we select the highest-accuracy configuration. This yields a compact, diverse set of pipelines from which a user can select their preferred configuration ($\theta_{selected}$).

Ensemble Deployment For the ensemble mode, the goal is to form a strong pool of candidates to maximize accuracy. To do this, we select all configurations from the evaluated pool whose execution accuracy is within a specified tolerance, ϵ, of the highest observed accuracy. This set of high-accuracy (but not necessarily diverse) configurations, $\{\theta_1, \ldots, \theta_n\}$, is then passed to Phase 2 for use in the ensemble.

5.2 Phase 2: Candidate Ensemble Selection

In the ensemble deployment mode, PRISM leverages the set of high-accuracy configurations curated during Phase 1. Each configuration θ_i is used to generate a candidate SQL query c_i, resulting in a candidate pool $\mathcal{C} = \{c_1, \ldots, c_n\}$. The goal is to select the candidate most likely to be correct.

One common strategy is *self-consistency*, where candidates are grouped based on their execution results, and a candidate SQL query from the largest group is returned. This approach assumes that the most consistent answer is the most reliable, which does not always hold. To address these limitations, prior work [33] employs a scoring strategy based on a *selection model*. Each pair of candidates (c_i, c_j) is compared using a fine-tuned binary classifier to determine which query better answers the user's question. Individual candidates accumulate votes based on how many comparisons they win, and the one with the highest score is selected. While effective, this approach can be sensitive to inconsistent selection model decisions and does not explicitly group semantically equivalent candidates, potentially leading to vote fragmentation among answers that are correct but syntactically different.

PRISM introduces an improved approach, *Grouped Pairwise Scoring*, detailed in Alg. 2. It builds on the selection model framework but incorporates two key enhancements:

- **Symmetric Agreement.** To filter out noisy comparisons, a vote is counted for candidate c_i over c_j only if the selection model prefers c_i in both comparison orders: (c_i, c_j) and (c_j, c_i).

Algorithm 3 Foreign-Key Guided Schema Filtering

Require: Original schema $\mathcal{S}$, depth limit D, FK follow prob p_{fk}, incoming FK prob p_{in}, non-key sampling range $[r_{\min}, r_{\max}]$
Output: Filtered schema S'

// Initialize BFS with a random starting table
Choose a random table T_0 from $\mathcal{S}$ with a primary key
Initialize queue $Q \leftarrow [(T_0, 0)]$
Initialize set of selected columns $C \leftarrow \text{PrimaryKeysOf}(T_0)$
Initialize set of visited tables $V \leftarrow \{T_0\}$
while Q is not empty **do**
    $(T, d) \leftarrow Q.\text{pop}()$
    if $d \geq D$ **then continue**
    end if
// Follow outgoing foreign keys
    for each FK column c in T **do**
        if random() $< p_{\text{fk}}$ and its target table T' is not in V **then**
            Add c and its PK target to C
            Q.append($(T', d+1)$)
            V.add(T')
        end if
    end for
// Follow incoming foreign keys
    for each PK column c in T **do**
        for each incoming FK from another table T'' **do**
            if random() $< p_{\text{in}}$ and T'' is not in V **then**
                Add c and the incoming FK column to C
                Q.append($(T'', d+1)$)
                V.add(T'')
            end if
        end for
    end for
end while
// Add a fraction of non-key columns from all visited tables
for each visited table T_v in V **do**
    Add a random fraction $r \in [r_{\min}, r_{\max}]$ of non-key columns in T_v to C
end for
$S' \leftarrow \text{ConstructSchemaFromColumns}(C)$
return S'

- **Execution Result Grouping.** Rather than scoring individual queries, we aggregate votes at the level of execution result groups. This ensures that semantically equivalent queries reinforce one another instead of splitting the vote.

The algorithm proceeds as follows. It first removes exact duplicate SQL queries from the pool. Then, for each ordered pair (c_i, c_j), the selection model determines which candidate better answers the input question q. This decision is based on (1) the union of database schema elements referenced by each candidate, (2) the natural language question q, and (3) the candidate queries along with their execution results. After all consistent, symmetric comparisons are complete, the execution result group with the highest cumulative score is selected, with ties broken by group size and then randomly. One representative candidate from this group is returned as the final output.

As shown in § 6.5, this approach leads to higher end-to-end accuracy than both self-consistency and prior selection-model-based approaches. Details of the selection model fine-tuning are provided in § 6.1.

5.3 Offline Synthetic Example Generation

Modern NL2SQL pipelines often benefit from many-shot, example-driven prompting. Prior work [7, 33] has shown that generating demonstrations at query time (*online synthetic example generation*) can significantly improve accuracy. These methods typically construct a diverse set of examples—some based on the full schema and others on filtered subsets—tailored to the user's query. However, online generation incurs high monetary cost due to multiple LLM calls per query.

To mitigate this, we propose an offline alternative that pre-generates diverse question-SQL pairs using schema variations. At inference time, relevant examples are retrieved via similarity search from this offline pool, eliminating the need for costly onlinegeneration. Our method involves two key steps: (1) *schema subset construction* and (2) *example store generation.*

1) Schema Subset Construction. We construct hundreds of schema subsets per database (e.g., 500 in our experiments) using three strategies:

- *(1) Random Column Sampling:* Uniformly samples a subset of columns from the full schema.
- *(2) Table-Based Sampling:* Samples a subset of tables, then selects a fraction of columns from each.
- *(3) Foreign-Key Guided Filtering:* this strategy (Alg. 3) traverses the schema graph by following foreign and primary key relationships. Starting from a random table, it expands outward via foreign key joins, up to a specified depth. For each visited table, a small fraction of non-key columns is also sampled to introduce diversity. This strategy ensures that the resulting schema subset captures meaningful join paths.

To further vary prompts, we randomly insert 0–3 categorical values per column in each subset.

2) Example Store Generation. For each unique schema subset, we prompt Gemini-pro-1.5 to generate 10 question-SQL pairs. The prompt includes the schema subset formatted as DDL and asks the LLM to create 10 queries and their corresponding SQL based solely on the presented schema. To eliminate syntactic errors, we ensure that every generated SQL query executes successfully on the target database. Next, we deduplicate the resulting pairs based on the question text to construct a final offline example pool for each database. While we do not perform additional semantic validation, our evaluation in § 6.8 demonstrates that this strategy is effective in practice.

Benefits. This approach decouples the expensive generation step from runtime inference. It also enables high-coverage, diverse many-shot examples that better generalize across varying query structures. During inference, relevant examples can be retrieved efficiently from this pool using question similarity. We show in § 6.8 that this offline strategy achieves nearly identical peak accuracy to online generation while being substantially more cost-effective.

6 Evaluation

In this section, we answer the following research questions:

- How does PRISM compare to strong baselines in both the single-candidate and ensemble settings in terms of execution accuracy and cost (§ 6.2)?
- How does PRISM perform across databases with diverse schema complexity (§ 6.3)?
- What configuration patterns emerge among the top-performing pipelines selected by PRISM (§ 6.4)?
- How effective is our ensemble selection algorithm (§ 6.5)?
- Can PRISM generalize to open-source models (§ 6.6)?
- How do different Bayesian Optimization acquisition functions influence accuracy–cost trade-offs (§ 6.7)?
- How does offline example generation compare to online generation in accuracy and cost (§ 6.8)?

Table 3. Performance Metrics for Different Schema Linking Strategies.

Schema Linking Strategies	FPR	SLR
Per-Column	61.53	94.20
Hybrid Per-Column	64.59	95.00
Hierarchical	6.01	80.25
Hybrid Hierarchical	15.06	85.87
FullSchema	85.84	100.00

Note: FPR = False Positive Rate, SLR = Schema Linking Recall.

- How robust is PRISM's cost–accuracy optimization under constraints on the quantity or quality of labeled data (§ 6.9)?
- How well does PRISM generalize to databases with complex schema (§ 6.10)?

Note. During the course of our experiments, the Gemini-pro-1.5 API was deprecated. Experiments reported in §§ 6.9 and 6.10 were therefore conducted using the updated reasoning model Gemini-flash-2.5. The remainder of the experimental setup remains identical to that described in § 6.1.

6.1 Experimental Setup

Dataset and Split. We evaluate on the bird benchmark [24], a large cross-domain NL2SQL dataset containing 12,751 unique question–SQL pairs across 95 databases spanning 37 professional domains. These databases are designed to resemble real-world settings, with complex schemas and noisy data. Our experiments focus on the development (dev) set, which includes 1534 question–SQL pairs across 11 databases. We chose the dev set because these databases are well-established in the NL2SQL literature, enabling the community to better interpret the configurations and trade-offs PRISM discovers.

Since PRISM performs schema-specific optimization that requires a small labeled dataset from the target schema, we split the questions for each database into two halves: 50% for the labeled set used in Phase 1 optimization and 50% as a held-out test set. This split is stratified by question difficulty.

Metrics. The primary evaluation metrics used are:

- **Execution Accuracy (%):** Following the official metric defined by the BIRD benchmark [24], we compute execution accuracy by executing both the generated and ground-truth SQL queries and performing a set-based equivalence check on their results. A query is considered correct only if the set of tuples returned by the generated query is identical to the set of tuples returned by the ground-truth query.
- **Total Dollar Cost ($):** The cumulative cost of LLM API calls, computed over both input and output tokens, for generating the final SQL query. To ensure stable comparisons that are not affected by fluctuations in API pricing, we report relative costs in our figures. For each cost–accuracy plot, all costs are normalized by the cost of the most expensive baseline method shown in that figure.

In addition to overall execution accuracy and cost, we also evaluate the performance of the schema linking using the following metrics. Ground truth schema for each query is determined from the ground truth SQL in the benchmark.

- **False Positive Rate (FPR):** Calculated per query as the ratio of retrieved irrelevant schema columns (columns retrieved but not present in the ground truth SQL) to the total number of retrieved columns, averaged across the evaluation set.
- **Schema Linking Recall (SLR):** The fraction of queries for which all schema columns required by the ground truth SQL were successfully retrieved. This is crucial, as accurate downstream

Table 4. Single-candidate deployment. Test accuracy and median per-database cost savings for single-candidate configurations. The median saving is reported with its Interquartile Range (IQR), calculated relative to the LongContext baseline.

Method	Test Accuracy (%)	Cost Saving % (Median ± IQR)
Gemini-pro-1.5	65.06	82.36 ± 21.10
Gemini-flash-2.0	66.36	98.63 ± 1.63
CHESS$_{(IR+SS+CG)}$	66.88	77.42 ± 2.73
LongContext	67.14	(Baseline)
PRISM	**69.48**	91.70 ± 13.37

SQL generation often depends on retrieving all required columns. It is worth noting that a functionally correct query (e.g., using COUNT(*) instead of COUNT(id)) may still be generated even if a required column was missed.

Tab. 3 shows the FPR and SLR statistics averaged over 5 independent runs over the entire bird dev dataset.

Implementation Details. For all experiments, unless otherwise specified, the prompt context includes any domain-specific hints provided as part of the bird benchmark. We run the Bayesian Optimization process for 30 trials. For our configuration selection algorithm (Alg. 1), we use a default value of $k = 4$ and an accuracy threshold of $\epsilon = 5\%$.

The pairwise selection model (θ_p) used in our ensemble deployment (Alg. 2) is a fine-tuned Gemini-flash-2.0 binary classifier. To train this model, we ran PRISM's Bayesian Optimization on the BIRD train split to generate candidate SQL queries. Each candidate was labeled based on execution correctness, resulting in a dataset of 99k pairs, where each example contains one correct and one incorrect candidate. Following prior work [33], we mitigated positional bias by including both orderings of each pair in the training data. The resulting model achieves 75% accuracy on a held-out test set.

Baselines. We compare against three strong baselines in the *single-candidate* setting:

- **CHESS$_{(IR+SS+CG)}$ [41]** It consists of three phases: an information retrieval phase to gather relevant sample column values and descriptions, a schema selection phase, and a candidate generation phase with up to 3 retries. It employs a decomposition generation strategy, where the question is decomposed into sub-questions and their solutions are stitched together to form the final SQL query.
- **Gemini-pro-1.5 and Gemini-flash-2.0** It provides the full database schema, sample column values, instructions, and hints within the prompt context. It uses Gemini-pro-1.5 and Gemini-flash-2.0, respectively, for SQL generation, employing a basic generation strategy with self-correction attempted up to a maximum of 3 times if execution errors occur.
- **LongContext [7]** In addition to Gemini-pro-1.5, this baseline incorporates an online synthetic example generation phase. It generates two categories of in-context examples: (1) SQL queries illustrating common features such as joins, aggregations, and filters using the full schema, and (2) examples focused on the filtered schema relevant to the input question. These examples are generated dynamically at query time and appended to the prompt to improve schema understanding.

In the *ensemble* setting, we compare against **CHASE-SQL [33]**, a multi-generator ensemble framework that constructs 21 candidates SQL queries using three fixed pipeline configurations. It then uses a fine-tuned LLM as a selection model to perform pairwise comparisons between candidates. The candidate that wins the most comparisons is chosen as the final answer.

Table 5. Ensemble deployment performance. Test accuracy of different ensemble selection strategies applied to the candidate pools generated by CHASE-SQL and PRISM.

Ensemble Selection Strategy	Test Accuracy (%)
CHASE-SQL Candidate Pool (21 candidates)	
Self-Consistency	70.4
Grouped Pairwise Scoring	73.7
PRISM Candidate Pool ($\varepsilon = 11$, mean 22 candidates)	
Self-Consistency	72.3
Grouped Pairwise Scoring	74.9

6.2 Performance Comparison with Single and Ensemble Baselines

Single-Candidate Performance. We first evaluate the effectiveness of PRISM by comparing its best single-candidate configuration (`num_sql_candidates` =1) against the single-candidate baselines. For this evaluation, we select the configuration representing the highest accuracy point on the Pareto frontier generated by PRISM and evaluate it on the held-out test set.

The results, summarized in Tab. 4, show that PRISM outperforms all baselines in test accuracy, achieving 69.48% compared to 67.14% for the strongest baseline, LongContext. In addition to its superior accuracy, PRISM also delivers significant cost savings. To provide a robust measure of efficiency that is not skewed by outlier databases, the table reports the median per-database cost saving relative to the LongContext baseline. Our PRISM configuration achieves a median saving of 91.70% ± 13.37%. This result demonstrates that the configurations found by PRISM are not only more accurate but cost-effective.

Candidate Ensemble Selection Performance. To evaluate the quality of our candidate generation, we conduct a direct comparison between the candidate pools produced by PRISM and CHASE-SQL. We set $\varepsilon = 11$ to yield comparable pool sizes (mean 22 for PRISM vs. 21 for CHASE-SQL).

We compare the effectiveness of the two candidate pools under two selection strategies. As shown in Tab. 5, PRISM consistently outperforms CHASE-SQL under both *Grouped Pairwise Scoring* (74.9% vs. 73.7%) and model-free *Self-Consistency* (72.3% vs. 70.4%). These results demonstrate that the Phase 1 optimization process curates a superior candidate pool compared to the fixed pipelines used in prior work, thereby improving the accuracy of downstream candidate selection strategies.

6.3 Per-Database Performance Analysis

In this section, we analyze PRISM's performance on individual databases to highlight the need for schema-specific tuning. Fig. 7 plots the cost–accuracy trade-offs across databases, comparing static baselines with the Pareto frontier of single-candidate configurations identified by PRISM in Phase 1.

This schema-specific approach proves highly effective. On a majority of databases, including toxicology, codebase_community, and student_club, PRISM identifies configurations that strictly dominate all baselines, achieving higher accuracy at lower cost. This adaptability is essential, as the strongest fixed baseline varies across schemas. For example, on student_club, CHESS$_{(IR+SS+CG)}$ outperforms LongContext, whereas on european_football_2, LongContext is clearly stronger. PRISM successfully finds a better configuration in both cases, underscoring the limitations of one-size-fits-all strategies.

Our analysis also reveals important limitations. On european_football_2, PRISM discovers a high-accuracy configuration but fails to select a more cost-efficient alternative that was explored during the search. This exclusion occurred because the cheaper configuration had low training

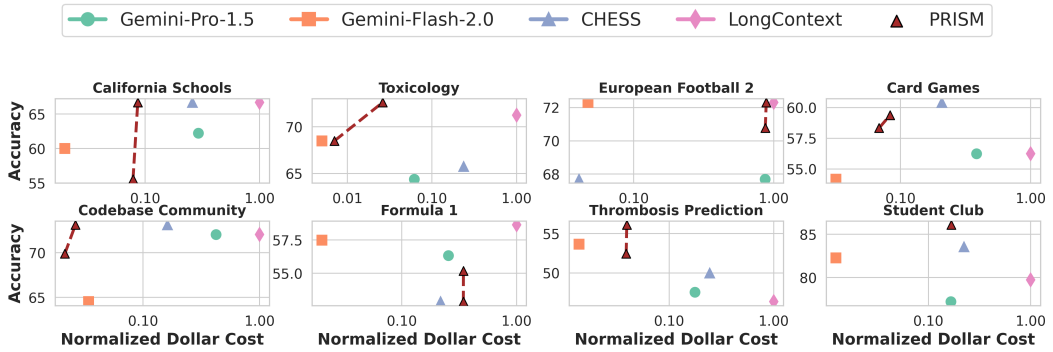


Fig. 7. Accuracy vs. cost trade-offs across databases. Each subplot shows test accuracy versus normalized dollar cost (log scale) for a specific database. Costs are normalized with respect to the LongContext baseline for each database. The brown dashed line represents the Pareto frontier of single-candidate configurations identified by PRISM in Phase 1.

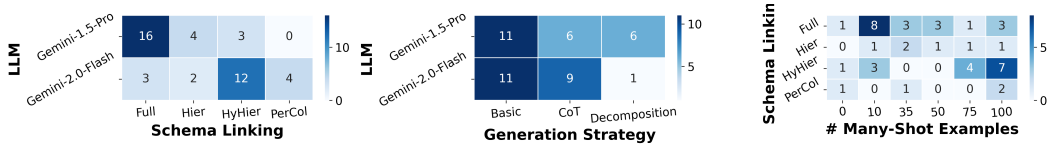


Fig. 8. Pairwise interaction analysis. Heatmaps display the frequency counts of co-occurring parameter values for selected configuration pairs, aggregated across the top-k configurations per database. Darker shades indicate higher co-occurrence frequency. Abbreviations: PerCol = Per-Column, Hier = Hierarchical, HyHier = Hybrid Hierarchical, Full = Full Schema.

accuracy and was filtered out for falling outside the 5% margin of the highest accuracy configuration. This reveals a deliberate trade-off in our heuristic, which favors configurations with demonstrated high performance during the optimization phase.

Finally, we observe generalization challenges on databases like card_games and formula_1, where some baselines outperform PRISM's selected configurations on the test set. These limitations suggest that, for certain schemas, larger or more representative training splits may be needed to improve generalization from training to test performance.

6.4 High-Performing Configuration Analysis

To better understand the characteristics of effective configurations, we analyze the top-k configurations selected by Phase 1 for each database. We select 4 configurations from each database, making a total of 44 configurations. This analysis focuses on identifying dominant patterns, configuration archetypes, and interactions among key parameters.

Diversity of High-Performing Configurations. We first assess the frequency of full configuration combinations among the top-k configurations. Even the most common configurations appear only 2–3 times across all databases, indicating high diversity and suggesting that optimal settings are highly schema-specific.

Configuration Archetypes via Clustering. To extract interpretable archetypes, we apply K-Prototypes clustering [16] (suitable for mixed-type data) to the 44 configurations. The resulting clusters reveal three dominant configuration types:

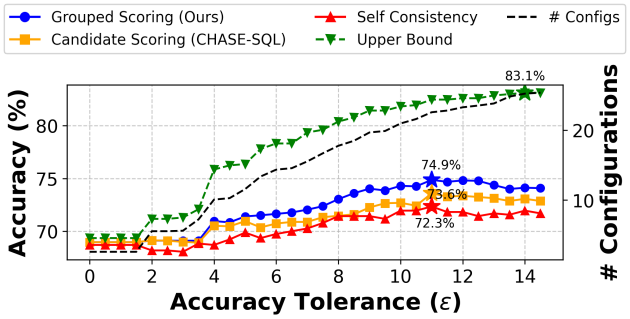


Fig. 9. Accuracy of different ensemble selection strategies as a function of accuracy tolerance (ε). The black dashed line indicates the number of configurations below each tolerance level (right y-axis).

- **Cluster 1 (size 19):** Hybrid Hierarchical, basic strategy, Gemini-flash-2.0, and high example count (median: 100).
- **Cluster 2 (size 15):** FullSchema, basic strategy, Gemini-flash-2.0, and low example count (median: 10).
- **Cluster 3 (size 10):** FullSchema, basic strategy, Gemini-pro-1.5, and medium example count (median: 35).

All clusters exhibit a consistent retry budget of 2. These clusters highlight that despite overall diversity, successful configurations align into a few representative archetypes.

Pairwise Parameter Interactions. We next explore parameter interactions via pairwise co-occurrence analysis (shown in Fig. 8), revealing several strong dependencies:

- **LLM vs. Schema Linking:** Gemini-pro-1.5 often co-occurs with FullSchema, while Gemini-flash-2.0 prefers Hybrid Hierarchical schema linking. This suggests that smaller models like Gemini-flash-2.0 benefit from receiving a pruned schema.
- **Schema Linking vs. Examples:** Hybrid Hierarchical, a pruning strategy, aligns with higher example counts (50–100), whereas FullSchema correlates with lower counts (0–10). This suggests a potential synergy: the added examples may compensate for information lost during schema pruning, allowing the model to benefit from a more concise schema.
- **Generation Strategy vs. LLM:** Both models favor the basic prompting strategy overall; however, decomposition is predominantly used with Gemini-pro-1.5, while CoT is more common with Gemini-flash-2.0. This aligns with model strengths: decomposition prompting involves reasoning over decomposed subproblems, a task better handled by Gemini-pro-1.5.

6.5 Ensemble Selection Strategies

Having established the superiority of the PRISM-generated candidate pool in § 6.2, we now evaluate different candidate selection strategies. We compare our Grouped Pairwise Scoring method (Alg. 2) against two strong baselines: (1) Candidate Scoring (CHASE-SQL) [33], which uses a fine-tuned model to perform pairwise comparisons, assign scores to individual candidates, and select the one with the highest score; and (2) Self-Consistency, a model-free baseline that groups candidates by their execution results and selects a representative from the largest group.

Fig. 9 shows the test accuracy of each strategy as a function of the accuracy tolerance ε, which controls the size of the candidate pool. We also include an Upper Bound, representing the ideal case where the system selects a correct SQL query if one exists in the pool. Grouped Pairwise Scoring consistently outperforms both baselines across all tolerance levels, achieving a peak accuracy of 74.9%, compared to 73.6% for Candidate Scoring and 72.3% for Self-Consistency. These improvements

Table 6. End-to-end single-candidate performance on the Qwen model family. PRISM is compared against the strongest fixed baseline for each model size. Cost savings are relative to the Qwen-14B-TCSL baseline.

Method	Test Accuracy (%)	Cost Saving %
Qwen-3B-TCSL	43.77	72.41
Qwen-7B-TCSL	60.13	49.46
Qwen-14B-TCSL	60.26	(Baseline)
PRISM	**64.94**	**13.87**

stem from our algorithm's design, which leverages group-level scoring and filters inconsistent pairwise comparisons to enhance selection robustness.

6.6 Generalization to Open-Source Models

To test the generality of our framework beyond Gemini models, we apply PRISM to the open-source Qwen models [11] (Qwen-14B, Qwen-7B, and Qwen-3B) across all 11 databases. We run Phase 1 (Bayesian Optimization) to identify configurations tailored to the unique architectural and contextual constraints of each Qwen model. For a fair comparison, we evaluate a fixed baseline configuration across all three models, consisting of the Hierarchical schema strategy, a basic prompting style, zero-shot examples, and a retry budget of 3. This setup reflects the practical constraint that Qwen's context window cannot support full-schema prompting, and mirrors settings shown to be effective in prior work [11].

Tab. 6 provides an end-to-end performance comparison of the best single-candidate configuration identified by PRISM against the strongest fixed baseline for each Qwen model size. The results confirm that PRISM's schema-specific optimization is effective. It achieves 64.94% test accuracy, a 4.68% improvement over the strongest fixed baseline (Qwen-14B-TCSL), while also delivering 13.87% in cost savings.

We next examine the configuration patterns that PRISM selects to adapt to the Qwen model family. Following the same methodology as our previous analysis § 6.4, we select the top-4 performing configurations for each of the 11 databases. Fig. 10 aggregates these 44 configurations and reveals that, unlike Gemini models, which often favor FullSchema or Hybrid Hierarchical, the Qwen models show a clear preference for the Hierarchical strategy. This is a natural adaptation to Qwen's smaller context window, which makes verbose prompts less feasible.

The corresponding interaction with many-shot examples also differs. Whereas Gemini models often compensate for schema pruning with high example counts (*e.g.*, 50–100), Qwen models generally perform best with fewer examples (typically 0–35), again conserving limited context space.

Apart from these general trends, our results show PRISM demonstrating a nuanced, schema-aware adaptation. On a smaller schema like toxicology (11 cols), PRISM correctly identifies that FullSchema is viable. For student_club, a moderately larger schema (48 cols), it selects Hierarchical schema linking paired with 100 examples - a strategy that mirrors the pattern observed in Gemini models as well. However, for much larger schemas such as european_football_2 (199 cols), it selects Hierarchical schema linking combined with low example counts (0–35). This flexibility underscores PRISM's ability to adapt to the specific interplay between model capacity and database schema complexity.

Fig. 11 further confirms that PRISM discovers high-performing configurations in the cost–accuracy space. We present results on four representative databases and show that the Pareto frontier (triangles) discovered by PRISM consistently outperforms the static baselines.

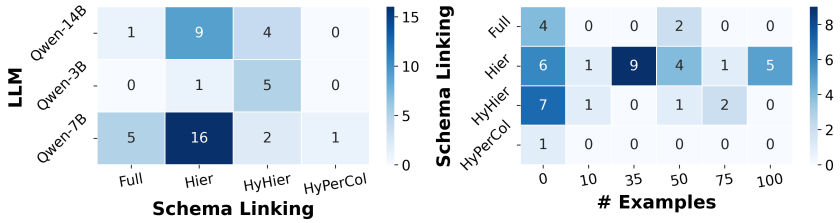


Fig. 10. Pairwise interaction analysis for the Qwen model family, aggregated across 11 databases. Abbreviations: HyPerCol = Hybrid Per-Column, Hier = Hierarchical, HyHier = Hybrid Hierarchical, Full = Full Schema.

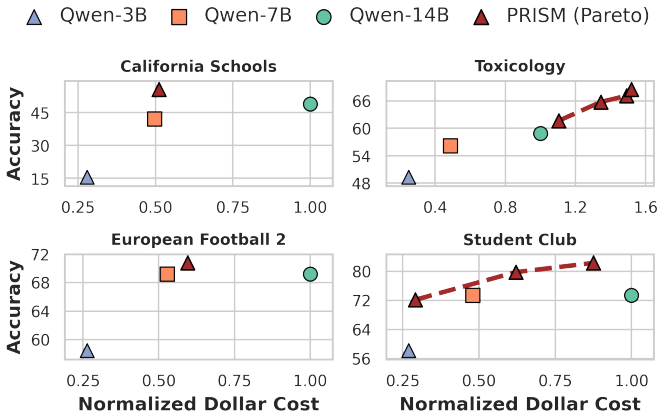


Fig. 11. Normalized cost–accuracy trade-offs for the Qwen model family. Each subplot shows PRISM's Pareto frontier (triangles) and the fixed baselines for four representative databases, with costs normalized per database with respect to the Qwen-14B baseline configuration.

Table 7. Median accuracy (± IQR) and percent cost saved during Phase 1 across three databases. % Optimization Cost Saved is relative to the most expensive acquisition strategy for each database.

Database	Acquisition Function	Accuracy (Median ± IQR)	% Optimization Cost Saved
codebase_community	EI	65.59 ± 1.08	0.0
	EICool	**65.59 ± 1.08**	41.7
	CEI	**65.59 ± 1.0**	27.7
european_football_2	EI	75.00 ± 0.00	6.6
	EICool	73.44 ± 0.78	33.2
	CEI	**78.12 ± 1.56**	0.0
superhero	EI	**89.06 ± 0.00**	0.0
	EICool	**89.06 ± 0.78**	29.5
	CEI	**89.06 ± 0.78**	8.3

Together, these findings confirm that PRISM operates as a robust framework, capable of automatically tailoring NL2SQL pipelines to the distinct capabilities of models.

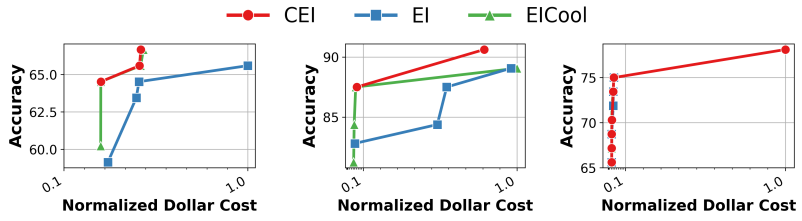


Fig. 12. Aggregated Pareto frontiers comparing configuration cost vs. accuracy across 3 databases. Each point represents a configuration explored by one of the acquisition functions during Phase 1. Costs are normalized per database and shown on a log scale.

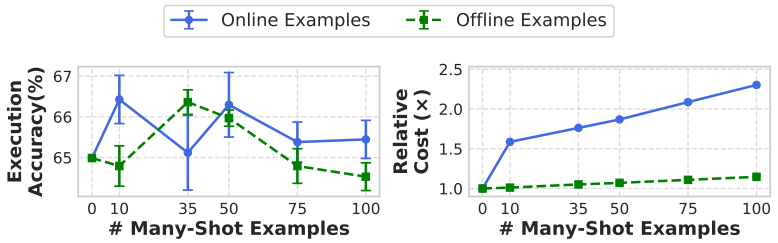


Fig. 13. Comparison of offline vs. online examples as a function of the many-shot examples. Relative cost is normalized to the cost at $n = 0$. Error bars indicate standard deviation.

6.7 Impact of Acquisition Function

We compare three acquisition functions in Phase 1, EI, EICool, and CEI —across three representative databases using five random seeds. We chose european_football_2 in particular because it has the largest number of columns and revealed several interesting configuration patterns. Tab. 7 and Fig. 12 show that CEI consistently identifies high-accuracy configurations while exploring a broader range of configuration costs. For example, on european_football_2, CEI achieves a median accuracy of 78.12%, outperforming both EI (75.00%) and EICool (73.44%). In codebase_community, it identifies cost-efficient configurations that EI misses due to narrow high-cost exploration.

While EI prioritizes accuracy, it can overcommit to expensive regions favored by the surrogate model. EICool strongly prefers low-cost options but struggles when high-accuracy configurations lie outside that regime. In contrast, CEI selects top candidates based on expected improvement and then filters them using cost-aware diversity, leading to more balanced configuration sets.

We caution against over-interpreting these trends: much of CEI's advantage appears to be empirical rather than theoretically grounded. Nonetheless, across the benchmark, it offers a practical trade-off between accuracy and cost, avoiding some of the failure modes observed in EI and EICool.

6.8 Offline vs. Online Many-Shot Examples

We compare our *offline* many-shot example strategy (§ 5.3) to a query-specific *online* strategy, adapted from prior work [7, 33]. While online examples are generated at query time using the full and pruned schemas, offline examples are retrieved from a pre-generated pool using similarity search. Both methods are evaluated on top of the Gemini-flash-2.0 baseline, varying the number of examples (0, 10, 35, 50, 75, 100) and aggregating over 5 runs.

As shown in Fig. 13, the offline strategy reaches a peak accuracy of 66.36% at n=35, closely matching the online strategy's 66.42% at n=10. However, the cost of the online strategy increases sharply with more examples, while the offline strategy remains comparatively flat, resulting in 35% lower relative cost at peak accuracy. The online strategy performs better at small n due to query-specific tailoring, but both methods degrade beyond $n > 50$, likely due to context saturation in Gemini-flash-2.0, where too many examples may introduce noise rather than useful signal.

These results validate the offline strategy as a cost-effective alternative to online generation, achieving comparable accuracy with significantly lower cost.

6.9 Robustness to Noisy and Limited Data

This subsection evaluates the robustness of PRISM when the quality or quantity of labeled data is constrained. We study two settings: (1) automatically generated (noisy) data obtained from SQL query logs, and (2) limited labeled data available for offline optimization.

Noisy Labeled Data. To assess the robustness of PRISM in scenarios where high-quality labeled NL–SQL pairs are not readily available, we evaluate its performance when trained on automatically generated (noisy) supervision. Existing systems (*e.g.*, [42, 43], Google BigQuery [12]) leverage chat histories and query logs to construct NL-SQL pairs, thereby improving the accuracy of subsequent queries. To emulate this realistic setting, we use only the SQL queries from the bird benchmark (mimicking query logs) and automatically synthesize the corresponding natural language questions using an LLM.

To generate natural language questions from SQL, we adopt the algorithm presented in OmniSQL [22]. It introduces linguistic diversity in generated questions by varying the generation style. For each question, the algorithm randomly selects one of the nine generation style (formal, colloquial, imperative, interrogative, descriptive, concise, vague, conversational, and metaphorical). Note that we excluded multi-term conversational style from our setting. Additionally, each generation prompt includes the ground-truth schema inferred from the SQL query. We refer readers to OmniSQL for further implementation details.

We then use the resulting noisy NL–SQL pairs as the labeled dataset in PRISM's offline optimization phase. On the held-out test set, the strongest fixed baseline (LongContext) achieves 66.6% execution accuracy. PRISM trained with ground-truth (clean) supervision achieves 70.8% accuracy with 92% cost savings relative to the baseline, while PRISM trained with noisy supervision achieves 69.1% accuracy with 70% cost savings. These results demonstrate that PRISM remains robust even when trained on automatically generated, noisy examples, supporting its practical applicability in settings where manually labeled data is scarce.

Limited Labeled Data. To assess data efficiency, we restrict the number of annotated ground-truth NL–SQL pairs per database to a maximum of 50 and rerun the offline optimization phase. Even under this constraint, PRISM maintains strong performance, achieving 69% test accuracy while preserving 92% of the median cost savings relative to the LongContext baseline (66.6% execution accuracy). These results confirm that PRISM remains effective even with modest annotation budgets.

6.10 Generalization to Complex Schemas

We evaluate PRISM's generalization on the BEAVER benchmark [4], which has a more complex schema than bird. We focus on the `DW` split, the only sub-database containing a sufficient number of queries for meaningful analysis. This split includes 97 tables and 1530 columns and provides neither column or table descriptions nor primary–foreign key relationships, making schema interpretation highly challenging.

The strongest baseline configuration on BEAVER uses Gemini-flash-2.5 with the FullSchema strategy and a basic prompting style, achieving an execution accuracy of 26.67%. The next best configuration, using Gemini-flash-2.0 with identical other parameters, achieves 8.88%. For reference, the original BEAVER paper reports that the best 1-shot execution accuracy using GPT-4o is 4.2%.

PRISM successfully identifies the same configuration as the best baseline, also achieving 26.67% execution accuracy. We found that none of the schema-linking strategies perform well on this schema, with the highest schema-linking recall (SLR) being only 0.34 for Hybrid Per-Column, which is precisely why PRISM's optimization correctly converged on the FullSchema. Since only a few correct SQL candidates could be generated for each question, the ensemble accuracy was also capped at 26.67%.

In conclusion, PRISM adapts effectively to metadata-poor, complex schemas. Further accuracy gains may require expanding PRISM's search space with new techniques and enriching schemas with metadata such as column and table descriptions, directions left for future work.

7 Discussion

Query-Level Specialization. While PRISM adapts configurations at the schema level, queries within the same schema can vary in complexity, motivating per-query specialization. Prior LLM routing work [5, 17, 28, 31] selects a model per query, but in NL2SQL, the optimal configuration involves multiple interacting parameters beyond just the LLM. EllieSQL [54] routes among three fixed pipelines but does not explore the broader configuration space. Extending PRISM for query-level routing over richer configurations is a promising direction.

Latency as an Optimization Objective. Latency is a critical objective for interactive NL2SQL applications and does not always correlate with cost. For instance, ensembles and Per-Column can invoke LLMs concurrently, reducing latency at higher cost. Modeling latency accurately requires accounting for inference time, API concurrency limits, and retrieval overheads.

Generalization of Offline Optimization. A practical concern is whether a configuration selected during the offline phase might overfit the tuning data, thus failing to generalize to online deployment. We acknowledge this potential risk; however, our problem structure mitigates it. The configuration search space, while large, is highly structured and discrete, with a finite number of choices for each component (*e.g.*, 3 prompting strategies, 5 schema linking methods). This space is fundamentally more constrained than the continuous parameter space of a traditional machine learning model. Consequently, we expect overfitting effects to be limited. This expectation of good generalization is validated by our empirical results. As shown in Tab. 4 and Fig. 7, configurations discovered by PRISM show strong performance on the held-out test set.

Cold-Start Scenarios. While PRISM provides schema-specific optimization benefits, its offline phase requires a small labeled dataset, which may not exist in true cold-start settings. In such cases a practitioner can initially deploy a strong, fixed configuration, such as one of the baselines evaluated in our study. This initial step mirrors the status quo for deploying a fixed NL2SQL pipeline. As query logs and chat histories accumulate over time, PRISM's offline optimization can be run to find a superior, schema-specific configuration to replace the initial baseline.

8 Related Work

NL2SQL and In-Context Learning. Early efforts in NL2SQL relied on handcrafted rules and templates [52], but the field progressed rapidly with the introduction of sequence-to-sequence neural models [39, 44], which enabled end-to-end mapping of natural language queries to SQL. Several models were proposed to improve schema understanding and compositional reasoning. IRNet [14] used LSTMs with self-attention for joint encoding of queries and schema. RAT-SQL [46]

and RASAT [36] enhanced representation with relation-aware self-attention. GNN-based models like SADGA [2] and LGESQL [3] further improved schema-query alignment.

Recent research has shifted toward leveraging large language models (LLMs) for NL2SQL. Zero-shot prompting using LLMs demonstrated early promise [37], followed by methods like DIN-SQL [34], DAIL-SQL [10], MAC-SQL [45], and C3 [9], which introduced in-context learning (ICL) and task decomposition. Fine-tuned open-source LLMs [11, 23, 35] have also emerged, although they still leverage ICL to further boost performance.

Multi-generator ensemble techniques [11, 20, 33, 41] generate many SQL candidates and use self-consistency or fine-tuned selector models to retrieve the most accurate candidate. These techniques often incur significant computational costs due to repeated LLM calls. In contrast, PRISM tackles these challenges directly by performing a cost-aware, schema-specific search to identify a set of individually strong configurations. This enables deployment either as a single cost-efficient pipeline or as a high-accuracy ensemble, depending on application needs.

Prompt Tuning and Workflow Optimization. Prompt and workflow optimization has gained increasing attention. Methods like OPRO [50], Agent Symbolic Learning [53], TextGrad [51], and Trace [6] use LLM-generated feedback or textual backpropagation for iterative refinement. Other frameworks, such as DSPy [19], program structured GenAI workflows and apply techniques like few-shot prompting and CoT injection. They frequently use variants of OPRO for optimization. GPTSwarm [55], in contrast, uses reinforcement learning to update DAG workflows but typically requires large-scale interaction budgets.

Our work is more closely aligned with BO-driven strategies like Cognify [15], which explores GenAI workflow autotuning under budget constraints. Similar goals also arise in systems like Palimpzest [25], which balances accuracy and cost when answering declarative queries over unstructured data, and DocETL [38], which uses agent-driven decomposition to improve document processing quality. However, unlike these general-purpose or loosely structured settings, we focus on the structured, domain-specific space of NL2SQL. Here, the challenge lies in configuring discrete, interdependent components, such as schema linking methods, prompting strategies, LLMs, and number of examples. PRISM applies cost-aware Bayesian Optimization to efficiently explore this space and surface high-performing configurations.

9 Conclusion

PRISM is a framework for systematically optimizing NL2SQL pipelines by navigating the trade-off between execution accuracy and monetary cost. Unlike prior work with fixed pipelines, PRISM formulates NL2SQL tuning as a schema-specific, multi-objective optimization problem. It follows a two-phase, optimize-then-deploy strategy: in the offline phase, it uses cost-aware Bayesian Optimization to identify a diverse pool of high-performing configurations; in the online phase, it supports both single-candidate and ensemble deployment using these configurations.

On the BIRD benchmark, PRISM achieves 69.48% execution accuracy in the single-candidate setting—a 2.34% gain over the strongest baseline—while reducing cost by 92%. In ensemble mode, accuracy rises to 74.9%. These results highlight the effectiveness of schema-aware tuning and validate PRISM as a cost-efficient and adaptable solution for NL2SQL deployment.

10 Acknowledgments

This work was supported in part by the U.S. National Science Foundation (IIS-2238431, IIS-2335881), Google, Cisco, Dolby, and Adobe. We thank colleagues in Georgia Tech Database Group for their constructive feedback in improving the system.

References

[1] Dean Allemang and Juan Sequeda. 2024. Increasing the LLM Accuracy for Question Answering: Ontologies to the Rescue! *arXiv preprint arXiv:2405.11706* (2024).

[2] Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. 2021. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems* 34 (2021), 7664–7676.

[3] Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. 2021. LGESQL: line graph enhanced text-to-SQL model with mixed local and non-local relations. *arXiv preprint arXiv:2106.01093* (2021).

[4] Peter Baile Chen, Fabian Wenz, Yi Zhang, Devin Yang, Justin Choi, Nesime Tatbul, Michael Cafarella, Çağatay Demiralp, and Michael Stonebraker. 2024. BEAVER: an enterprise benchmark for text-to-sql. *arXiv preprint arXiv:2409.02038* (2024).

[5] Shuhao Chen, Weisen Jiang, Baijiong Lin, James Kwok, and Yu Zhang. 2024. Routerdc: Query-based router by dual contrastive learning for assembling large language models. *Advances in Neural Information Processing Systems* 37 (2024), 66305–66328.

[6] Ching-An Cheng, Allen Nie, and Adith Swaminathan. 2024. Trace is the next autodiff: Generative optimization with rich feedback, execution traces, and llms. *arXiv preprint arXiv:2406.16218* (2024).

[7] Yeounoh Chung, Gaurav T Kakkar, Yu Gan, Brenton Milne, and Fatma Ozcan. 2025. Is Long Context All You Need? Leveraging LLM's Extended Context for NL2SQL. *arXiv preprint arXiv:2501.12372* (2025).

[8] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. 2004. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*. 253–262.

[9] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306* (2023).

[10] 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* (2023).

[11] Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, et al. 2024. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599* (2024).

[12] Google Cloud. 2025. NL2SQL with BigQuery and Gemini. https://cloud.google.com/blog/products/data-analytics/nl2sql-with-bigquery-and-gemini. Accessed: 2025-09-03.

[13] Gauthier Guinet, Valerio Perrone, and Cédric Archambeau. 2020. Pareto-efficient acquisition functions for cost-aware Bayesian optimization. *arXiv preprint arXiv:2011.11456* (2020).

[14] 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* (2019).

[15] Zijian He, Reyna Abhyankar, Vikranth Srivatsa, and Yiying Zhang. 2025. Cognify: Supercharging Gen-AI Workflows With Hierarchical Autotuning. *arXiv preprint arXiv:2502.08056* (2025).

[16] Zhexue Huang. 1998. Extensions to the k-means algorithm for clustering large data sets with categorical values. *Data mining and knowledge discovery* 2, 3 (1998), 283–304.

[17] Saehan Jo and Immanuel Trummer. 2025. SpareLLM: Automatically Selecting Task-Specific Minimum-Cost Large Language Models under Equivalence Constraint. *Proceedings of the ACM on Management of Data* 3, 3 (2025), 1–26.

[18] Donald R Jones, Matthias Schonlau, and William J Welch. 1998. Efficient global optimization of expensive black-box functions. *Journal of Global optimization* 13 (1998), 455–492.

[19] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan, Saiful Haq, Ashutosh Sharma, Thomas T Joshi, Hanna Moazam, et al. 2023. Dspy: Compiling declarative language model calls into self-improving pipelines. *arXiv preprint arXiv:2310.03714* (2023).

[20] Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv preprint arXiv:2405.07467* (2024).

[21] Eric Hans Lee, Valerio Perrone, Cedric Archambeau, and Matthias Seeger. 2020. Cost-aware Bayesian optimization. *arXiv preprint arXiv:2003.10870* (2020).

[22] Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. OmniSQL: Synthesizing High-Quality Text-to-SQL Data at Scale. *Proc. VLDB Endow.* 18, 11 (Sept. 2025), 4695–4709. doi:10.14778/3749646.3749723

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

[24] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2023. 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 (2023), 42330–42357.

[25] Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baile Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, Rana Shahout, et al. 2025. Palimpzest: Optimizing ai-powered analytics with declarative query

processing. In *Proceedings of the Conference on Innovative Database Research (CIDR)*. 2.
[26] Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. 2024. A Survey of NL2SQL with Large Language Models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109* (2024).
[27] Xiping Liu and Zhao Tan. 2023. Divide and prompt: Chain of thought prompting for text-to-sql. *arXiv preprint arXiv:2304.11556* (2023).
[28] Keming Lu, Hongyi Yuan, Runji Lin, Junyang Lin, Zheng Yuan, Chang Zhou, and Jingren Zhou. 2023. Routing to the expert: Efficient reward-guided ensemble of large language models. *arXiv preprint arXiv:2311.08692* (2023).
[29] 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* (2024).
[30] Jonas Mockus. 1998. The application of Bayesian methods for seeking the extremum. *Towards global optimization* 2 (1998), 117.
[31] Isaac Ong, Amjad Almahairi, Vincent Wu, Wei-Lin Chiang, Tianhao Wu, Joseph E Gonzalez, M Waleed Kadous, and Ion Stoica. 2024. Routellm: Learning to route llms with preference data. *arXiv preprint arXiv:2406.18665* (2024).
[32] Yoshihiko Ozaki, Yuki Tanigaki, Shuhei Watanabe, and Masaki Onishi. 2020. Multiobjective tree-structured parzen estimator for computationally expensive optimization problems. In *Proceedings of the 2020 genetic and evolutionary computation conference*. 533–541.
[33] 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* (2024).
[34] Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems* 36 (2024).
[35] Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117* (2024).
[36] Jiexing Qi, Jingyao Tang, Ziwei He, Xiangpeng Wan, Yu Cheng, Chenghu Zhou, Xinbing Wang, Quanshi Zhang, and Zhouhan Lin. 2022. Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql. *arXiv preprint arXiv:2205.06983* (2022).
[37] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498* (2022).
[38] Shreya Shankar, Tristan Chambers, Tarak Shah, Aditya G Parameswaran, and Eugene Wu. 2024. Docetl: Agentic query rewriting and evaluation for complex document processing. *arXiv preprint arXiv:2410.12189* (2024).
[39] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).
[40] 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* (2023).
[41] 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* (2024).
[42] Matthias Urban, Jialin Ding, David Kernert, Kapil Vaidya, and Tim Kraska. 2025. Utilizing Past User Feedback for More Accurate Text-to-SQL. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics (HILDA '25)*. Association for Computing Machinery, New York, NY, USA, Article 10, 7 pages. doi:10.1145/3736733.3736739
[43] Kapil Vaidya, Jialin Ding, Sebastian Kosak, David Kernert, Chuan Lei, Xiao Qin, Abhinav Tripathy, Ramesh Balan, Balakrishnan Narayanaswamy, and Tim Kraska. 2025. TailorSQL: An NL2SQL System Tailored to Your Query Workload. *arXiv preprint arXiv:2505.23039* (2025).
[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
[45] 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* (2023).
[46] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942* (2019).
[47] Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171* (2022).
[48] 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 (2022), 24824–24837.
[49] Christopher KI Williams and Carl Edward Rasmussen. 2006. *Gaussian processes for machine learning*. Vol. 2. MIT press Cambridge, MA.

[50] Chengrun Yang, Xuezhi Wang, Yifeng Lu, Hanxiao Liu, Quoc V Le, Denny Zhou, and Xinyun Chen. 2023. Large language models as optimizers. *arXiv preprint arXiv:2309.03409* (2023).
[51] Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. 2024. TextGrad: Automatic "Differentiation" via Text. arXiv:2406.07496 [cs.CL] https://arxiv.org/abs/2406.07496
[52] John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*. 1050–1055.
[53] Wangchunshu Zhou, Yixin Ou, Shengwei Ding, Long Li, Jialong Wu, Tiannan Wang, Jiamin Chen, Shuai Wang, Xiaohua Xu, Ningyu Zhang, Huajun Chen, and Yuchen Eleanor Jiang. 2024. Symbolic Learning Enables Self-Evolving Agents. arXiv:2406.18532 [cs.CL] https://arxiv.org/abs/2406.18532
[54] Yizhang Zhu, Runzhi Jiang, Boyan Li, Nan Tang, and Yuyu Luo. 2025. Elliesql: Cost-efficient text-to-sql with complexity-aware routing. *arXiv preprint arXiv:2503.22402* (2025).
[55] Mingchen Zhuge, Wenyi Wang, Louis Kirsch, Francesco Faccio, Dmitrii Khizbullin, and Jürgen Schmidhuber. 2024. Gptswarm: Language agents as optimizable graphs. In *Forty-first International Conference on Machine Learning*.

Received July 2025; revised October 2025; accepted November 2025