# Distill-C: Enhanced NL2SQL via Distilled Customization with LLMs

**Cong Duy Vu Hoang**[1*], **Gioacchino Tangari**[2*], **Clemence Lanfranchi**[3*],
**Dalu Guo**[2], **Paul Cayet**[3], **Steve Siu**[2], **Don Dharmasiri**[2], **Yuan-Fang Li**[2],
**Long Duong**[2], **Damien Hilloulin**[3], **Rhicheek Patra**[3], **Sungpack Hong**[3], **Hassan Chafi**[3]
[1]Oracle Analytics Cloud (OAC), Australia
[2]Oracle Health & AI (OHAI), Australia
[3]Oracle Labs, Switzerland
{vu.hoang, gioacchino.tangari, clemence.lanfranchi}@oracle.com

## Abstract

The growing adoption of large language models (LLMs) in business applications has amplified interest in Natural Language to SQL (NL2SQL) solutions, in which there is competing demand for high performance and efficiency. Domain- and customer-specific requirements further complicate the problem. To address this conundrum, we introduce Distill-C, a distilled customization framework tailored for NL2SQL tasks. Distill-C utilizes large teacher LLMs to produce high-quality synthetic data through a robust and scalable pipeline. Fine-tuning smaller and open-source LLMs on this synthesized data enables them to rival or outperform teacher models an order of magnitude larger. Evaluated on multiple challenging benchmarks,[1] Distill-C achieves an average improvement of 36% in execution accuracy compared to the base models from three distinct LLM families. Additionally, on three internal customer benchmarks, Distill-C demonstrates a 22.6% performance improvement over the base models. Our results demonstrate that Distill-C is an effective, high-performing and generalizable approach for deploying lightweight yet powerful NL2SQL models, delivering exceptional accuracies while maintaining low computational cost.

## 1 Introduction

The increasing capabilities of large language models (LLMs) have led to their growing integration into business environments for streamlining routine tasks (Minaee et al., 2024; Liu et al., 2024). A key application is NL2SQL (Natural Language to SQL) translation, where developers frequently need to generate SQL queries for diverse business use cases (Zhu et al., 2024). Although state-of-the-art LLMs achieve high performance on public benchmarks, their large resource and computational demands, coupled with performance limitations in certain real-world contexts, make smaller specialized models a more suitable option for many practical applications. However, smaller LLMs often underperform relative to their larger counterparts, limiting their practical effectiveness in demanding scenarios.

One of the primary motivations for this work is the emerging area of NL2SQL data synthesis and knowledge distillation. Existing research has explored approaches to data synthesis and distillation for NL2SQL applications, yet these methods remain generalized rather than tailored to the specific needs of real-world customer environments. In recent work (Yang et al., 2024a) propose a "SQLer" model that generates training examples across diverse topics and domains. However, this approach does not tailor the distillation process to specific business applications. Similarly, another study (Chen et al., 2023) introduced personalized distillation for code generation by addressing small-model code execution errors, though it is not extended to NL2SQL.

We propose Distill-C (**Distill**ed **C**ustomization), a novel framework for NL2SQL distillation that introduces customizable elements to address specific customer use cases, requirements, and expectations. Distill-C leverages teacher LLMs to generate distilled knowledge, which is then transferred to smaller student models. By incorporating customized synthesis techniques, error-driven reference examples, and tailored distillation strategies, our approach enhances the accuracy and resource efficiency of smaller NL2SQL models, making them more practical for real-world applications.

Our contributions feature a scalable pipeline with the following key components:

- **Customization**: Integrates customer-specific features into the data synthesis for high-quality NL2SQL data.

[*]Equal contributions & corresponding authors
[1]Datasets are available at https://github.com/ClemenceLanfranchi/Distill-C



*Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Industry Track)*, pages 833–848

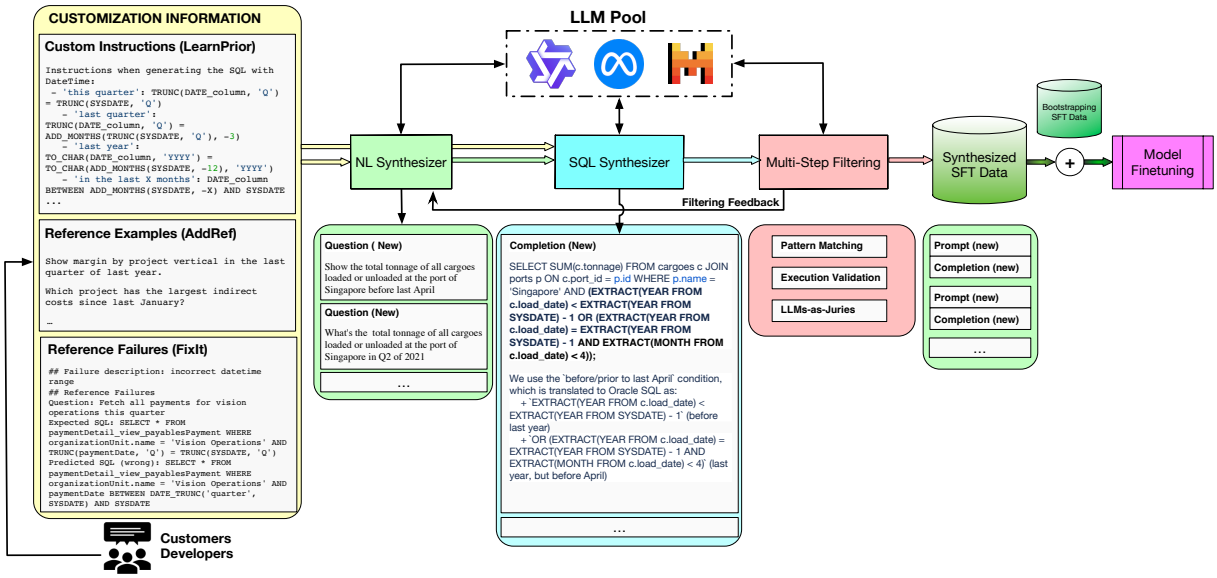



Figure 1: **The Proposed Distill-C Framework.**

- **Targeted Distillation**: Utilizes an ensemble of LLMs to balance their strengths and weaknesses, generating tailored datasets with features like date-time handling, financial analytics, and SQL compliance.
- **Modular Synthesis**: Separates natural language and SQL synthesis, leveraging multiple LLMs for better data diversity and robustness.
- **Quality Assurance**: Uses a multi-step filtering process (pattern matching, execution checks, LLM juries) to refine data quality.

Our Distill-C framework effectively enables small LLMs to perform on par with, or even surpass, their teacher models, exhibiting gains of 36% on average across different families of models and on various challenging benchmarks.

## 2 Methodology

### 2.1 Customization Scenarios

We present three distinct scenarios, including **AddRef**, **LearnPrior**, and **FixIt** - each of which is based on a reasonable assumption often confirmed in enterprise settings, where product and engineering teams typically have the capacity to provide a few examples, instructional guidance, and error feedback from early model deployments.

**AddRef: Incorporating Reference Examples.** Reference examples consist of a pre-defined subset of natural language (NL) queries provided by the **Customer** and serve as a basis for guiding data generation by LLMs. It is essential that these generated NL examples not only closely resemble the reference examples but also exceed them in complexity and originality.

**LearnPrior: Leveraging Prior Custom Instructions.** The **Customer** provides a limited set of statements detailing prior requirements and expectations for SQL responses generated by NL2SQL models. These statements convey the **Customer's** insights into how model outputs should align with their specific needs.

**FixIt: Distilling Targeted Knowledge from Error Feedback.** In this scenario, the **Customer** has initial access to a baseline model that is evaluated to identify a set of unacceptable model errors. These errors serve as starting points for bootstrapping targeted improvements, helping the model avoid similar issues in subsequent iterations.

### 2.2 The Distill-C Framework

We developed our **Distill**ed **C**ustomization framework, abbreviated as **Distill-C**, to synthesize tailored knowledge specifically adapted to the customer scenarios described above. The core components of our proposed **Distill-C** framework are illustrated in Figure 1. The framework comprises distinct NL and SQL synthesizers, followed by a three-stage filtering pipeline, and it enables the integration of knowledge from multiple advanced LLMs at each stage.

#### 2.2.1 Distillation Pipelines

Our framework decouples NL and SQL synthesis, which, though less resource-efficient than

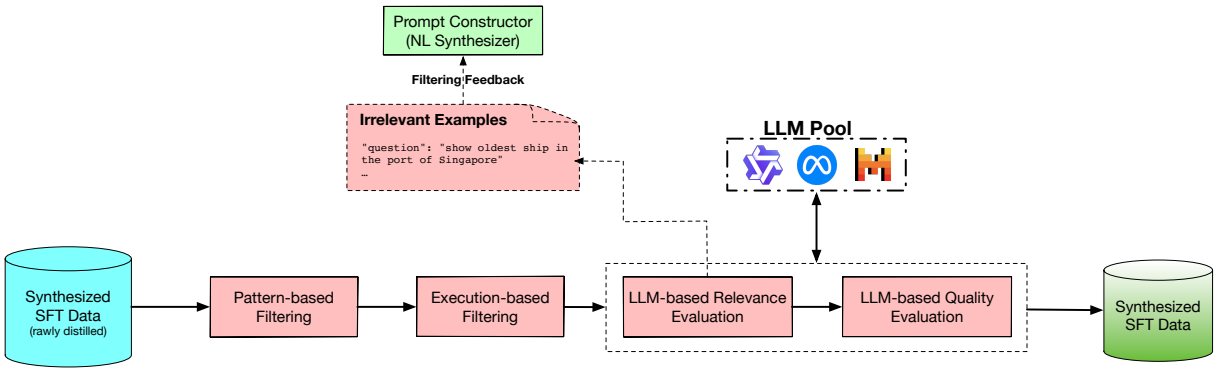



Figure 2: The Multi-Step Filtering Pipeline in our **Distill-C** Framework.

a single-step approach, offers two key benefits: *First*, independent generation by different LLMs enhances data diversity; *Second*, it leverages model-specific strengths. For example, while Llama3.1-70B-Instruct excels at generating realistic queries for a database schema, it may miss OracleSQL-specific nuances better addressed by Mixtral-8x22B-Instruct-v0.1, as shown in Table 1.

**NL Synthesizer Pipeline.** The NL synthesizer produces new NL queries or questions, guided by the customer's customization scenarios, including reference NL examples[2] (AddRef); prior expert instructions (LearnPrior); and targeted knowledge from error feedback (FixIt). These scenarios can be applied individually or in combination.

The NL synthesis process[3] begins with **Reference NL Extraction & Sampler**, where NL queries are sampled from reference examples, balancing inspiration with diversity within the LLM's context window. The **Prompt Constructor** then assembles NL generator prompts by combining these sampled NL examples and a database (DB) schema.[4] We also utilize discarded examples from previous generation rounds, incorporating a limited selection of them into the prompt as negative examples, which helps to iteratively refine the natural language synthesis process.

Finally, multiple LLMs (preferably 50B+ parameters) generate diverse NL queries by leveraging high-temperature sampling and varied random seeds, benefiting from their superior instruction following and generation diversity.[5] The outcome of the NL synthesis phase is a set of new NL queries relevant to the customer use case, each mapped to a DB schema.

**SQL Synthesizer Pipeline.** Starting with a set of {NL question, DB schema} pairs generated in earlier steps, the SQL synthesizer employs multiple Generator LLMs to translate each question into its corresponding SQL query. This process produces a preliminary, or "raw", distillation dataset (prior to filtering), where each entry forms a complete NL2SQL data point pairing the DB schema and NL question as the prompt with the SQL query as the completion. This dataset serves as a foundation for transferring knowledge from strong foundational LLMs into smaller models.

Key aspects of the SQL synthesis process (detailed in Appendix Figure 6) include:

- **Diverse LLMs as Generators**: Multiple LLMs enhance data diversity and address model-specific gaps, with some excelling in constructs like the Oracle SQL dialect.
- **Instruction-Conditioned Generation**: Task-specific instructions (LearnPrior) ensure SQL outputs align with customer requirements, including handling complex datetime structures (intervals, absolute and relative references).[6]

The synthesis process includes three key steps:

1. **Prompt Constructor**: Combines user queries, database schemas, and task-specific instructions to create effective prompts.
2. **SQL Generation**: LLMs generate SQL queries with descriptions, forming a synthetic supervised fine-tuning (SFT) dataset that clarifies complex SQL elements.
3. **Prompt Post-Processing**: Strips instructions from prompts in the SFT dataset to ensure

[2] consisting of 100 examples or fewer to initiate the data synthesis process.

[3] as further illustrated in Appendix Figure 5.

[4] sampled from a pool of training DB schemas.

[5] Despite their capability, proprietary LLMs (OpenAI; Anthropic; Gemini) are excluded from this process due to licensing restrictions on production use of their generated data.

[6] as further illustrated in Appendix Figure 8.

| Model Variant | DateTime (%) | | | | Financial Analytics (%) | | OracleSQL Compliance (%) | |
|---|---|---|---|---|---|---|---|---|
| | spd-ora | spd-lite | bd-lite | bd-ora | spd+bd-ora | spd+bd-lite | spd-ora | bd-ora |
| **Student LLMs & SFT with Distill-C (A-Full Setting)** | | | | | | | | |
| CodeQwen1.5-7B-Chat | 30.4 | 58.1 | 37.9 | 2.6 | 24.8 | 47.8 | 33.9 | 4.6 |
| +Distill-C (A-Full) | **74.0** | **68.7** | **57.2** | **33.8**† | **89.5**† | **84.1**† | **77.6** | **34.8**† |
| Llama-3.1-8B-Instruct | 29.8 | 62.6 | 41.3 | 2.6 | 17.0 | 35.9 | 36.1 | 3.1 |
| +Distill-C (A-Full) | **81.2**† | **67.6** | **59.3** | **29.5** | **83.2** | **78.2** | **79.4**† | **32.0** |
| Mistral-7B-Instruct-v0.3 | 22.1 | 46.4 | 22.2 | 2.6 | 21.1 | 24.5 | 38.4 | 4.4 |
| +Distill-C (A-Full) | **74.6** | **65.4** | **38.8** | **31.2** | **84.5** | **80.4** | **77.3** | **28.2** |
| **Out-of-the-Box Strong LLMs (selected)** | | | | | | | | |
| Qwen2-72B-Instruct (teacher) | 32.0 | 67.0 | 55.7 | 8.1 | 41.2 | 62.1 | 42.4 | 9.0 |
| Llama-3.1-70B-Instruct (teacher) | 24.3 | 62.0 | **61.6**† | 4.3 | 1.6 | 42.3 | 34.4 | 4.4 |
| Mixtral-8x22B-Instruct-v0.1 (teacher) | 48.6 | 64.8 | 42.0 | 21.4 | 67.5 | 71.3 | 54.1 | 16.9 |
| Mistral-Large-Instruct-2407 | 51.4 | **73.7**† | 53.9 | 16.2 | 83.6 | 83.2 | 58.1 | 20.6 |
| DeepSeek-Coder-V2-Instruct | 44.2 | 71.5 | 55.3 | 15.0 | 65.2 | 78.2 | 53.8 | 19.4 |

Table 1: **Task performances on DateTime, Financial Analytics, and OracleSQL Compliance.** †marks column bests; bold shows Distill-C induced performance. Notations: spd: Spider, bd: Bird, ora: OracleSQL, lite: SQLite.

smaller models learn directly from distilled examples.

#### 2.2.2 Multi-Step Filtering Pipeline

The training examples derived from the NL & SQL Synthesizer pipelines, consisting of (i) a prompt with a new question and (ii) an SQL completion, undergo a multi-step filtering process, as illustrated in Figure 2, to ensure data quality and minimize noise:

- **Pattern-Based Filtering**: Removes examples with non-target syntax (e.g., MySQL-specific keywords for Oracle SQL), reducing the load on resource-intensive downstream filters.
- **Execution-Based Filtering**: Validates SQL by executing it on real databases linked to schema contexts, discarding non-executable queries to prevent negatively impacting model performance.
- **LLM-Based Quality Evaluation**: Uses multiple strong LLMs as "juries" (Verga et al., 2024) to evaluate and rank examples for semantic accuracy to ensure alignment with intended NL meaning. This automated approach replaces manual review for large datasets.
- **LLM-Based Relevance Evaluation**: Ensures examples are relevant to the target use case by requiring unanimous agreement among LLMs. Irrelevant data is flagged as "Filtering Feedback" (Figure 1) for refining the NL synthesis.

#### 2.2.3 Finetuning

The final step involves finetuning the smaller target LLM using synthesized instruction data and a small bootstrapping dataset, which is crucial for mitigating biases and preventing model collapse (Gerstgrasser et al., 2024).

## 3 Experiments

### 3.1 Evaluation Tasks

We evaluate our approach on customer-identified tasks, including:

- **DateTime**: Generating SQL for complex temporal conditions, including relative (e.g., *"last 2 quarters"*) and composite clauses (e.g., *"first quarter of the last 5 years"*).
- **Financial Analytics**: Querying trends, correlations, and financial metric breakdowns (e.g., profits by country or quarter).
- **OracleSQL Compliance**: Producing syntactically correct OracleSQL queries.

### 3.2 Data and Evaluation Settings

**Experimental Data.** We built our experimental data using Spider (1.0) (Yu et al., 2018) and BIRD (Li et al., 2024a). For each task, we prepared three datasets: (i) a curated test set; (ii) a small development set for customization via AddRef, LearnPrior, and FixIt scenarios; (iii) a training set generated with the **Distill-C** pipeline. The training, testing and dev sets respectively comprise 199, 31, 10 disjoint DB schemas to prevent data leakage. Data statistics are in Table 3.

**Metric.** We use execution accuracy (Zhong et al., 2020) to evaluate our framework, which compares the execution results of the generated SQL query and the ground-truth on the corresponding database.

### 3.3 Model Settings

We evaluated our proposed Distill-C framework with a series of settings, progressing from NL-only (B) to complete (A-Full), which enables systematic evaluation of the impact of increasing supervision

| Customer | Use Case | Student Model | Distill-C Model | Distill-C Impact |
|---|---|---|---|---|
| **Customer 1** | Account payables and receivables management (4 schemas; 192/497 examples with datetime) | 80% | **97%** | Distill-C → DateTime |
| **Customer 2** | Information technology services and consulting (1 schema; 25/28 examples with financial analytics) | 54% | **78%** | Distill-C → Financial Analytics |
| **Customer 3** | Autonomous database (6 schemas; 99/99 examples with OracleSQL compliance) | 42% | **71%** | Distill-C → OracleSQL Compliance |

Table 2: Impact of Our Distill-C Method on Customer Benchmarks.

| Task | Origin | SQL Dialect | Train | Dev | Test |
|---|---|---|---|---|---|
| **DateTime** | Bird | OracleSQL | 9,621 | 115 | 533 |
| | Bird | SQLite | 33,173 | 78 | 234 |
| | Spider | OracleSQL | 13,460 | 131 | 680 |
| | Spider | SQLite | 37,172 | 97 | 179 |
| **Financial** | Bird | OracleSQL | 13,460 | 63 | 1,753 |
| **Analytics** | Bird | SQLite | 23,091 | 113 | 734 |
| | Spider | OracleSQL | 17,734 | 108 | 3,820 |
| | Spider | SQLite | 35,749 | 123 | 1,366 |
| **OracleSQL** | Bird | OracleSQL | 29,877 | 319 | 1,469 |
| **Compliance** | Spider | OracleSQL | 39,369 | 326 | 1,478 |

Table 3: Statistics of Train, Dev, and Test Datasets.

| Setting | Description |
|---|---|
| **B** | Distill-C w/ AddRef (NL): Uses 10 to 100 NL-only examples for data synthesis without SQL supervision. |
| **C** | Distill-C w/ AddRef (NL) + LearnPrior: Adds tailored instructions to NL-only examples to guide SQL generation. |
| **D** | Distill-C w/ AddRef (NL+SQL): Adds SQL supervision with paired NL + SQL examples for explicit NL-to-SQL mappings. |
| **E** | Distill-C w/ AddRef (NL) + LearnPrior + FixIt: Extends **C** with incorrect SQL examples to train error recognition. |
| **A-Full** | Full Distill-C: AddRef (NL+SQL) + LearnPrior + FixIt |

Table 4: Summary of evaluation settings.

and tailored training signals on model performance, as shown in Table 4. The distillation signals from teacher LLMs are derived in Table 5.

### 3.4 Public Main Results

The experimental results in Table 1 highlight the effectiveness of our proposed **Distill-C** framework, which integrates three customization scenarios (AddRef, LearnPrior, FixIt) to enhance the performance of various student LLMs across three challenging tasks: DateTime, Financial Analytics, and Oracle SQL Compliance. Our approach achieves significant performance gains across three foundational LLMs: CodeQwen1.5-7B-Chat (26.2%, 55.5%, 36.9%), Llama-3.1-8B-Instruct (25.3%, 54.3%, 36.1%), and Mistral-7B-Instruct-v0.3 (29.2%, 59.7%, 31.4%) for DateTime, Financial Analytics, and OracleSQL Compliance, respectively. These improvements across multiple benchmarks underscore the robustness of our method in enhancing LLM capabilities across diverse tasks.

| Student LLM | Teacher LLM(s) |
|---|---|
| Qwen1.5-7B-Instruct | Qwen2-72B-Instruct, Mixtral-8x22B-Instruct-v0.1 |
| Llama3.1-8B-Instruct | Llama3.1-70B-Instruct, Mixtral-8x22B-Instruct-v0.1 |
| Mistral-7B-Instruct-v0.3 | Mixtral-8x22B-Instruct-v0.1, Llama3.1-70B-Instruct |

Table 5: Student & Teacher LLMs used for distillation.

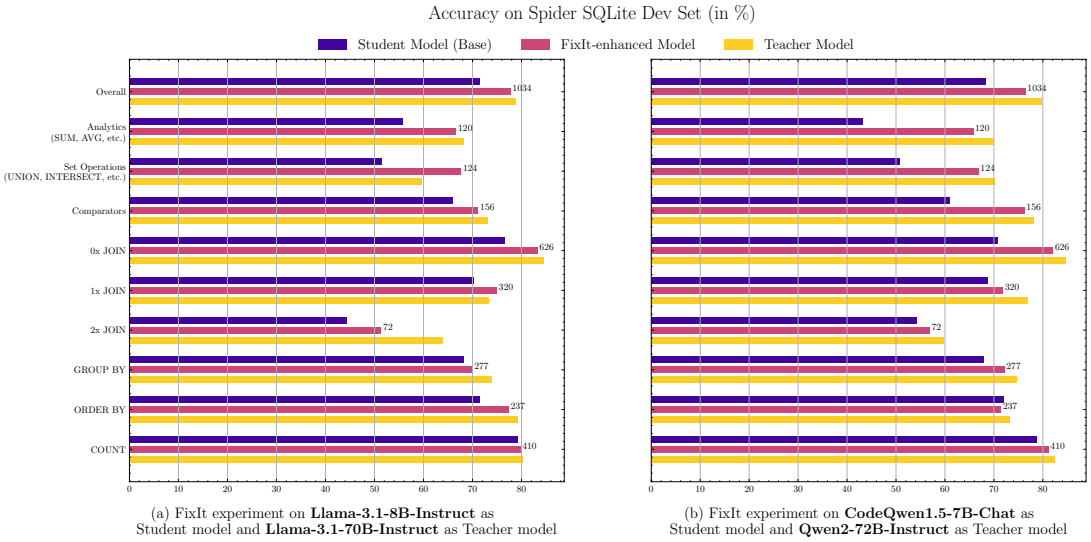
Furthermore, the distilled models surpass several strong out-of-the-box LLMs, including their teacher models such as Qwen2-72B-Instruct, Llama-3.1-70B-Instruct, and Mixtral-8x22b-Instruct-v0.1, which can be attributed to the tailored prompts that are used to guide the data synthesis process, fostering better SQL generation from the teacher models. Our fine-tuned models outperform larger state-of-the-art LLMs (e.g., Mistral-Large-Instruct-2407 and DeepSeek-Coder-V2-Instruct) on multiple benchmarks, showcasing the effectiveness of the Distill-C framework. These findings demonstrate the potential of the Distill-C framework to significantly enhance smaller LLMs, enabling them to handle complex tasks more effectively while providing substantial efficiency benefits for deployment.

### 3.5 Customer Impact

We demonstrated the business impact of our Distill-C method through enhanced performance gains on internal and customer-specific datasets.[7]

The performance boost of Distill-C on domain-specific tasks, as shown in Table 2, highlights its capability to address key challenges in customer-specific tasks such as DateTime handling, financial analytics, and SQL compliance, improving average accuracy significantly, by 22.6 absolute points. For DateTime tasks in Customer 1's account management use case, Distill-C achieved near-perfect accuracy (97%), demonstrating its robustness in handling temporal data critical for financial workflows. In Customer 2's financial analytics use case, the model significantly improved performance from

[7]Due to proprietary restrictions, we are unable to disclose the specifics of the customer schemas as well as benchmark sets for the NL2SQL tasks.




(a) FixIt experiment on **Llama-3.1-8B-Instruct** as Student model and **Llama-3.1-70B-Instruct** as Teacher model

(b) FixIt experiment on **CodeQwen1.5-7B-Chat** as Student model and **Qwen2-72B-Instruct** as Teacher model

Figure 3: **FixIt Ablation Study Experiments.** Performance of student models finetuned with the FixIt scenario using Distill-C on Spider (dev) sub-groups, showing results for student, finetuned, and teacher models, with sample counts per group.

54% to 78%, showcasing its ability to handle complex financial datasets and provide actionable insights. Finally, for Customer 3, focused on OracleSQL compliance in autonomous database use case, Distill-C delivered a substantial gain, raising accuracy from 42% to 71%. These results underscore Distill-C's versatility and effectiveness in enhancing precision and reliability across specialized tasks in diverse domains.

### 3.6 Ablation Study

We conduct two ablation studies to assess the impact of individual scenarios in Distill-C.

**Individual FixIt Scenario.** We evaluate the FixIt scenario using Llama-3.1-8B-Instruct and CodeQwen1.5-7B-Chat as student LLMs. Errors identified from the Spider training set (Yu et al., 2018) are processed through our data generation pipeline, where the NL Prompt Constructor (Figure 5) utilize these errors to guide teacher LLMs (Qwen2-72B-Instruct for CodeQwen and Llama-3.1-70B-Instruct for Llama) to create targeted datasets used to finetune the student models, producing FixIt-enhanced versions.

On the Spider development set, FixIt achieves performance improvements of 6.4% and 8% for Llama-3.1-8B-Instruct and CodeQwen1.5-7B-Chat, respectively, significantly narrowing gaps with their teacher models. Figure 3 shows notable gains in Analytics and Set Operations, effectively addressing key weaknesses.

**Full Scenarios.** Figure 4 demonstrates the consistent and substantial improvements achieved by integrating all scenarios (AddRef+LearnPrior+FixIt) within our Distill-C framework. While the AddRef scenario alone (Setting B) already brings a significant improvement of 24.7% on average, showcasing the importance of finetuning models on tasks that are similar to the target tasks, we also see that providing prior knowledge and leveraging errors is key to obtaining optimal performance. Moreover, the similarity in performance between scenarios C and D (respectively +30.4% and +32.6% on average) tends to show that custom instructions and examples of ground truth SQL queries are both valid options to distill prior knowledge. This integration leads to significant performance gains across a diverse range of benchmarks, including DateTime, Financial Analytics, and Oracle SQL Compliance, showcasing the versatility and robustness of our approach. Notably, these improvements are observed consistently across multiple student LLMs, underscoring the generalizability and effectiveness of the proposed framework. Overall, the results highlight how the synergistic combination of these scenarios enables Distill-C to address complex challenges and deliver superior outcomes, making it a compelling solution for advancing language understanding and task-specific performance.

## 4 Related Work

Recent advancements in NL2SQL research have explored techniques to enhance the performance of

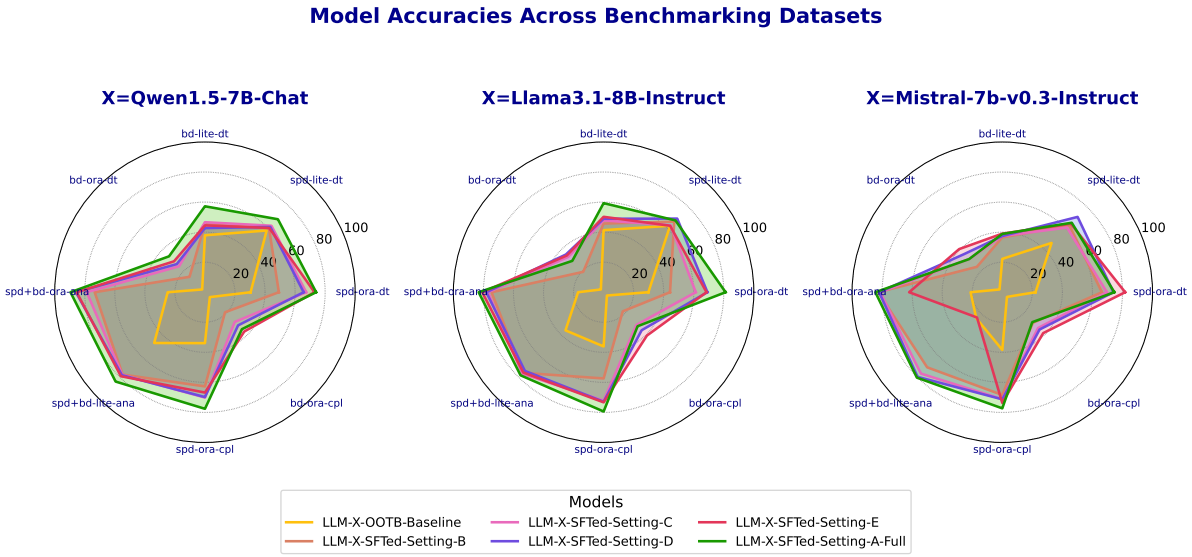



Figure 4: **Ablation study with distillation settings (Table 4).** Notations: spd: Spider, bd: Bird, dt: DateTime, ana: Analytics, ora: OracleSQL, lite: SQLite, cpl: Compliance. Numerical results are reported in Appendix B.3.

Large Language Models (LLMs).

**Prompt Engineering and Reasoning.** Prompt engineering has been explored to optimize NL2SQL capabilities of LLMs. PET-SQL (Li et al., 2024b) adopts a two-round framework with enhanced representations, and EPI-SQL (Liu and Tan, 2024) generates error-prevention prompts to reduce LLM errors. Self-correction and iterative refinement have also been explored in SQL-CRAFT (Xia et al., 2024) and DART-SQL (Mao et al., 2024), which integrate interactive feedback loops. However, these approaches are not well-suited to smaller Large Language Models (LLMs) because they necessitate acute reasoning capabilities that such models typically lack. On the other hand, Distill-C addresses this limitation by focusing on bridging the performance gap between large and small LLMs. This method leverages the advanced reasoning abilities of larger LLMs to distill their knowledge into more compact forms, thereby enhancing the capabilities of smaller models without requiring extensive computational resources.

**Synthetic Data Generation.** Recent works have shown the great promise of synthetic data. SQL-GEN (Pourreza et al., 2024) produces dialect-specific synthetic training data, while SENSE (Yang et al., 2024b) utilizes synthetic data for domain generalization and preference learning. Our approach focuses on creating tailored datasets that cater to specific customer needs by integrating targeted instructions and relevant examples into our data generation pipeline. Unlike previous work, we further customize the data generation process for individual student language models (LLMs) using error-driven reference examples.

## 5 Conclusion

We introduce Distill-C, a novel customizable distillation framework for enhancing small LLMs in NL2SQL tasks for enterprise applications. Despite their smaller sizes, the enhanced models by Distill-C achieve significant gains over strong baselines across benchmarks, including DateTime, Financial Analytics, and Oracle SQL Compliance. The initial costs associated with Distill-C, which involve hosting larger LLMs for data generation and fine-tuning smaller models, are offset by long-term advantages. These benefits arise because business units can then utilize more efficient and specialized smaller LLMs, ultimately leading to a substantial return on investment. Our work lays the foundation for robust distillation solutions, enabling the development of specialized NL2SQL models that can be tailored to specific business needs.

Our future work will explore extensions to preference alignment training (Rafailov et al., 2024) and applications to other practical tasks.

## Acknowledgments

We extend our sincere appreciation to our colleagues at the Science Team within Oracle Cloud Infrastructure (OCI) for their support and valuable feedback.

We are grateful to Giulia Carocari for her assistance in translating SQL queries between SQLite and Oracle SQL, as well as the anonymous reviewers whose valuable feedback significantly improved this work.

## References

Hailin Chen, Amrita Saha, Steven Hoi, and Shafiq Joty. 2023. Personalized distillation: Empowering open-sourced LLMs with adaptive learning for code generation. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 6737–6749, Singapore. Association for Computational Linguistics.

Matthias Gerstgrasser, Rylan Schaeffer, Apratim Dey, Rafael Rafailov, Henry Sleight, John Hughes, Tomasz Korbak, Rajashree Agrawal, Dhruv Pai, Andrey Gromov, Daniel A. Roberts, Diyi Yang, David L. Donoho, and Sanmi Koyejo. 2024. Is model collapse inevitable? breaking the curse of recursion by accumulating real and synthetic data. *Preprint*, arXiv:2404.01413.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C.C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2024a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. 2024b. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency. *Preprint*, arXiv:2403.09732.

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? *Preprint*, arXiv:2408.05109.

Xiping Liu and Zhao Tan. 2024. Epi-sql: Enhancing text-to-sql translation with error-prevention instructions. *Preprint*, arXiv:2404.14453.

Toby Mao. 2024. Sqlglot: Python sql parser and transpiler. `https://github.com/tobymao/sqlglot`. Accessed: 2024-11-29.

Wenxin Mao, Ruiqi Wang, Jiyu Guo, Jichuan Zeng, Cuiyun Gao, Peiyi Han, and Chuanyi Liu. 2024. Enhancing text-to-SQL parsing through question rewriting and execution-guided refinement. In *Findings of the Association for Computational Linguistics ACL 2024*, pages 2009–2024, Bangkok, Thailand and virtual meeting. Association for Computational Linguistics.

Shervin Minaee, Tomas Mikolov, Narjes Nikzad, Meysam Chenaghlu, Richard Socher, Xavier Amatriain, and Jianfeng Gao. 2024. Large language models: A survey. *Preprint*, arXiv:2402.06196.

Mohammadreza Pourreza, Ruoxi Sun, Hailong Li, Lesly Miculicich, Tomas Pfister, and Sercan O. Arik. 2024. Sql-gen: Bridging the dialect gap for text-to-sql via synthetic data and model merging. *Preprint*, arXiv:2408.12733.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct preference optimization: your language model is secretly a reward model. NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Pat Verga, Sebastian Hofstatter, Sophia Althammer, Yixuan Su, Aleksandra Piktus, Arkady Arkhangorodsky, Minjie Xu, Naomi White, and Patrick Lewis. 2024. Replacing judges with juries: Evaluating llm generations with a panel of diverse models. *Preprint*, arXiv:2404.18796.

Hanchen Xia, Feng Jiang, Naihao Deng, Cunxiang Wang, Guojiang Zhao, Rada Mihalcea, and Yue Zhang. 2024. $r^3$: "this is my sql, are you with me?" a consensus-based multi-agent system for text-to-sql tasks. *Preprint*, arXiv:2402.14851.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024a. Synthesizing text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875, Bangkok, Thailand. Association for Computational Linguistics.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024b. Synthesizing text-to-sql data from weak and strong llms. *Preprint*, arXiv:2408.03256.

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*.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online. Association for Computational Linguistics.

Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. 2024. Large language model enhanced text-to-sql generation: A survey. *Preprint*, arXiv:2410.06011.

## A Additional Figures

We include additional figures to illustrate the components of our Distill-C framework: the NL Synthesizer Pipeline in Figure 5 and the SQL Synthesizer Pipeline in Figure 6, respectively.

## B Additional Tables

### B.1 Experimental Setup: Training and Inference Configurations

We also provide our training and inference hyperparameter configurations in Table 8.

### B.2 Evaluation Tasks

In Table 6, we present detailed descriptions and examples of the three evaluation tasks used to assess the impact of our **Distill-C** framework.

### B.3 Ablation Study Evaluation

We further provide the detailed results of our ablation study (shown in Figure 4) in Table 7.

## C SQL Dialect Conversion

We utilize the SQLGlot library (Mao, 2024) to translate SQL queries from the Bird and Spider datasets from SQLite to the OracleSQL dialect. To enhance the translations, we apply a custom post-processor to address potential parsing issues and align with OracleSQL conventions.

## D Prompts in The Distill-C Framework

### D.1 Prompts for NL and SQL Synthesizer Pipelines

We also present additional prompt templates utilized across various components of our Distill-C framework, including:

- Figure 7 - An example prompt template for the NL Synthesizer pipeline (AddRef scenario).
- Figure 8 - An example prompt template for the SQL Synthesizer pipeline (LearnPrior scenario) with a focus on DateTime use case.

### D.2 Prompts for Multi-Step Filtering Pineline

Given the large scale of the Synthetic SFT Data (over 10,000 instances), manual or human-in-the-loop evaluation is not feasible. Therefore, we rely on soft evaluation using multiple strong LLMs as judges, following (Verga et al., 2024). We employed two primary evaluation phases as shown in Figure 2 as follows:

- **LLM-based Quality Evaluation.** In this evaluation, each 'judge' LLM assigns a 1-to-5 star score per criterion, with a cut-off as a hyperparameter: consensus on '5 stars' is required for SQL correctness and compliance, and at least '4 stars' for NL quality (Figure 9).
- **LLM-based Relevance Evaluation** This evaluation step queries multiple LLMs to assess the relevance of a generated example to the use case in the Reference Examples, using prompts in Figure 10. Examples marked 'relevant' by all LLMs are added to the final synthetic fine-tuning set, while those marked 'irrelevant' are stored as 'irrelevant examples' for the Input Schema to guide future NL generation (Figure 5).

| Task Name | Description | DB Schema | Sample NL Query | Sample OracleSQL Query |
|---|---|---|---|---|
| **DateTime** | Handling complex temporal conditions, including absolute, relative, and composite clauses. Absolute clauses use fixed dates, relative clauses involve SYSDATE, and composite clauses mix both. | wta_1 (Spider) | Get the ranking history of Serena Williams since March 2015. | `SELECT rankings.* FROM rankings JOIN players ON rankings.player_id = players.player_id WHERE players.first_name = 'Serena' AND players.last_name = 'Williams' AND TO_CHAR(rankings.ranking_date, 'YYYY-MM') >= '2015-03'` |
| | | financial (Bird) | Which client got his/her card issued since last May? Show the client ID. | `SELECT T2.client_id FROM "client" T1 INNER JOIN disp T2 ON T1.client_id = T2.client_id INNER JOIN card T3 ON T2.disp_id = T3.disp_id WHERE TRUNC(T3.issued, 'MM') >= ADD_MONTHS(TRUNC(SYSDATE - INTERVAL '1' YEAR, 'YYYY'), 4)` |
| **Financial Analytics** | Producing trends, correlations, and breakdown of financial metrics by date-time intervals and categories. Includes handling complex clauses like GROUP BY, ORDER BY, and Common Table Expressions (CTEs). | e_commerce (Spider) | What is the total revenue generated by each product for each customer in 2023, and which product generated the highest revenue for each customer? | `SELECT c.customer_id, c.customer_first_name, c.customer_last_name, p.product_id, p.product_name, SUM(p.product_price) AS total_revenue, RANK() OVER (PARTITION BY c.customer_id ORDER BY SUM(p.product_price) DESC) AS revenue_rank FROM Customers c JOIN Orders o ON c.customer_id = o.customer_id JOIN Order_Items oi ON o.order_id = oi.order_id JOIN Products p ON oi.product_id = p.product_id JOIN Shipments s ON o.order_id = s.order_id JOIN Invoices i ON s.invoice_number = i.invoice_number WHERE EXTRACT(YEAR FROM i.invoice_date) = 2023 GROUP BY c.customer_id, c.customer_first_name, c.customer_last_name, p.product_id, p.product_name ORDER BY c.customer_id, total_revenue DESC` |
| | | financial (Bird) | Calculate the total loans approved per district in 2023, broken down by status, sorted in descending order. | `SELECT d.district_id, d.A2 AS district_name, l.status, SUM(l.amount) AS total_loan_amount FROM district d JOIN "account" a ON d.district_id = a.district_id JOIN loan l ON a.account_id = l.account_id WHERE EXTRACT(YEAR FROM l."date") = 2023 GROUP BY d.district_id, d.A2, l.status ORDER BY total_loan_amount DESC` |
| **OracleSQL Compliance** | Handling OracleSQL-dialect syntax, including ORDER BY with "FETCH FIRST/LAST {N} ROWS", correct quoting, and casing for schema object names. | car_1 (Spider) | What are the different models created by either General Motors or over 3500 lbs? | `SELECT DISTINCT T1."model" FROM model_list T1 JOIN car_makers T2 ON T1.Maker = T2."id" JOIN car_names T3 ON T1."model" = T3."model" JOIN cars_data T4 ON T3.MakeId = T4."id" WHERE T2.FullName = 'General Motors' OR T4.Weight > 3500` |
| | | financial (Bird) | List out the accounts who have the earliest trading date in 1995 ? | `SELECT account_id FROM trans WHERE EXTRACT(YEAR FROM "date") = 1995 ORDER BY "date" ASC FETCH FIRST 1 ROWS ONLY` |

Table 6: Details of the Evaluation Tasks.

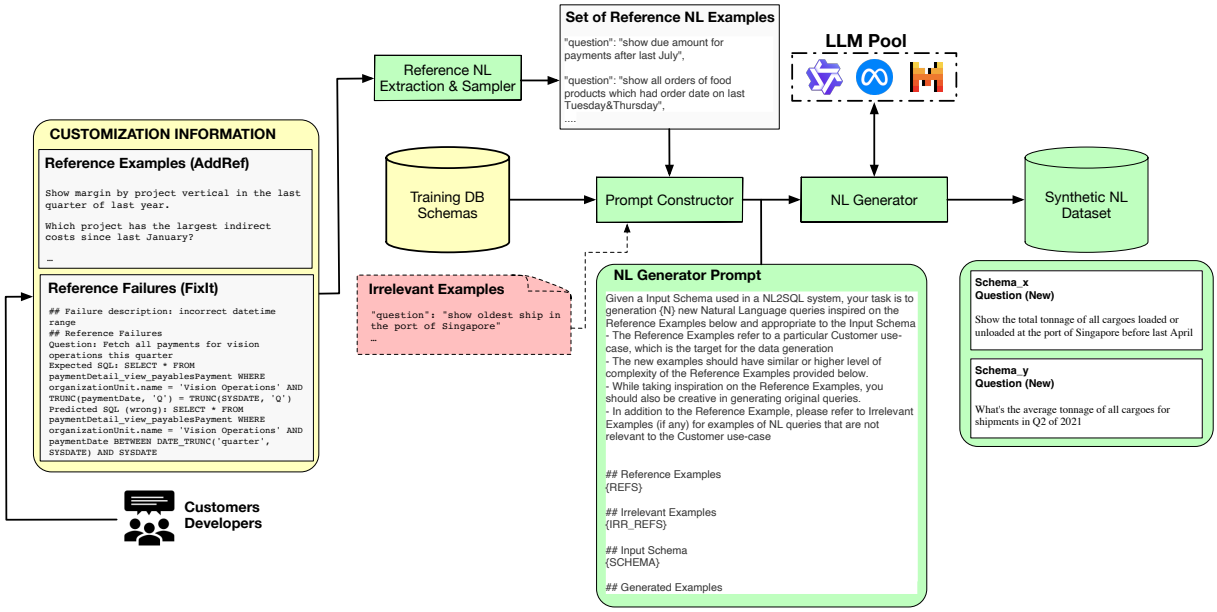



Figure 5: The NL Synthesizer Pipeline in our **Distill-C** Framework.

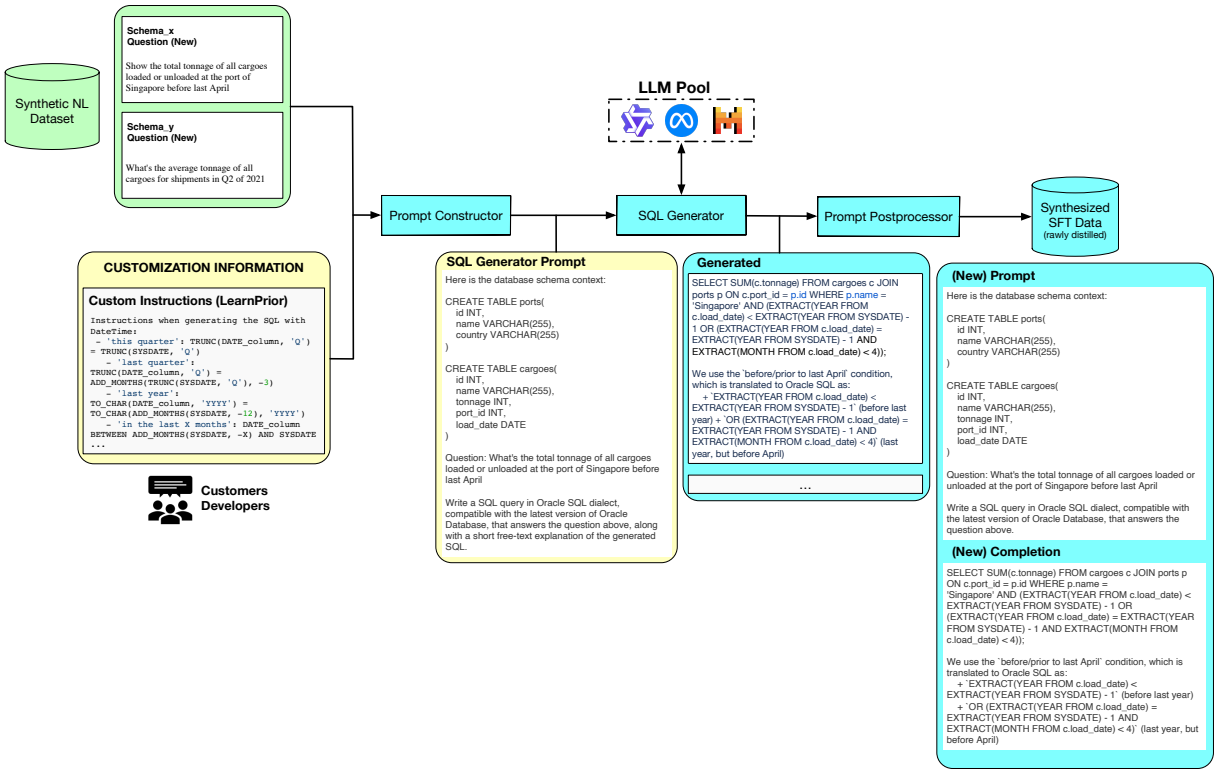



Figure 6: The SQL Synthesizer Pipeline in our **Distill-C** Framework.

| Model Variant | DateTime | | | | Financial Analytics | | OracleSQL Compliance (%) | |
|---|---|---|---|---|---|---|---|---|
| | spd-ora | spd-lite | bd-lite | bd-ora | spd+bd-ora | spd+bd-lite | spd-ora | bd-ora |
| **Qwen1.5-7B-Chat** | | | | | | | | |
| OOTB-Baseline | 30.4 | 58.1 | 37.9 | 2.6 | 24.8 | 47.8 | 33.9 | 4.6 |
| SFTed-Setting-B | 49.2 | 60.3 | 45.3 | 14.5 | 73.4 | 77.5 | 62.7 | 18.9 |
| SFTed-Setting-C | 67.4 | 62.6 | 46.6 | 24.4 | 78.7 | 77.8 | 70.1 | 28.0 |
| SFTed-Setting-D | 65.7 | 61.5 | 42.7 | 26.5 | 85.9 | 78.1 | 69.8 | 31.2 |
| SFTed-Setting-E | 72.9 | 60.5 | 44.5 | 29.1 | 85.4 | 79.1 | 66.8 | **37.0** |
| **SFTed-Setting-A-Full** | **74.0** | **68.7** | **57.2** | **33.8** | **89.5** | **84.1** | **77.6** | 34.8 |
| **Llama3.1-8B-Instruct** | | | | | | | | |
| OOTB-Baseline | 29.8 | 62.6 | 41.3 | 2.6 | 17.0 | 35.9 | 36.1 | 3.1 |
| SFTed-Setting-B | 44.2 | 66.5 | 45.7 | 19.2 | 74.6 | 75.9 | 57.4 | 18.0 |
| SFTed-Setting-C | 61.3 | 63.1 | 47.8 | 33.3 | 76.9 | 76.7 | 72.2 | 30.8 |
| SFTed-Setting-D | 69.1 | 69.3 | 48.9 | **35.5** | 78.1 | 74.1 | 72.9 | 35.8 |
| SFTed-Setting-E | 68.5 | 62.6 | 50.2 | 34.6 | 80.8 | 75.6 | 73.3 | **40.8** |
| **SFTed-Setting-A-Full** | **81.2** | **67.6** | **59.3** | 29.5 | **83.2** | **78.2** | **79.4** | 32.0 |
| **Mistral-7b-v0.3-Instruct** | | | | | | | | |
| OOTB-Baseline | 22.1 | 46.4 | 22.2 | 2.6 | 21.1 | 24.5 | 38.4 | 4.4 |
| SFTed-Setting-B | 66.3 | 63.1 | 36.5 | 23.9 | 82.6 | 70.7 | 68.4 | 28.8 |
| SFTed-Setting-C | 69.6 | 60.9 | 37.4 | 31.2 | 81.3 | 76.6 | 71.2 | 33.7 |
| SFTed-Setting-D | 74.0 | **70.9** | 37.4 | 35.9 | 82.0 | 79.7 | 71.3 | 35.1 |
| SFTed-Setting-E | **81.8** | 64.5 | **39.0** | **40.6** | 62.0 | 23.8 | 73.7 | **38.5** |
| **SFTed-Setting-A-Full** | 74.6 | 65.4 | 38.8 | 31.2 | **84.5** | **80.4** | **77.3** | 28.2 |

Table 7: **Performance comparison of model variants on DateTime, Financial Analytics, and OracleSQL tasks for the different distillation scenarios.** Notations: OOTB (Out-Of-The-Box), spd (Spider), bd (Bird), ora (OracleSQL), lite (SQLite).

| | **Finetuning Configuration** |
|---|---|
| **Pretrained Checkpoints** | CodeQwen1.5-7B-Chat, Llama3.1-8B-Instruct, Mistral-7B-Instruct-v0.3 |
| **Batch Size** | 512 examples per step |
| **Learning Rate** | 1e-6 (with linear decay) |
| **Warmup Steps** | 2,000 |
| **Max Sequence Length** | 8192 tokens |
| **Optimizer** | Paged AdamW 8-bit ($\beta_1 = 0.9$, $\beta_2 = 0.95$) |
| **Weight Decay** | N/A |
| **Gradient Clipping** | 1.0 |
| **Training Steps** | 20,000 |
| **Evaluation Metrics** | Checkpoint-based Execution Accuracy |
| **Hardware Setup** | 8 NVIDIA A100 40GB GPUs |
| | **Inference Configuration** |
| **Decoding Strategy** | Random Sampling |
| **Temperature** | 0.5 |
| **Top-k Sampling** | 40 |
| **Top-p Sampling** | 0.9 |
| **Max Sequence Length** | 2048 tokens |
| **Batch Size** | 32 |

Table 8: Configuration details for training and inference in our experiments.

Prompt Example for NL Synthesizer Pipeline (AddRef)

```
Given a Input Schema used in a NL2SQL system, your task is to generation 5 new Natural
    Language queries inspired on the Reference Examples below and appropriate to the Input
    Schema
- The Reference Examples refer to a particular Customer use-case, which is the target for the
    data generation
- The new examples should have similar or higher level of complexity of the Reference
    Examples provided below.
- While taking inspiration on the Reference Examples, you should also be creative in
    generating original queries.
- In addition to the Reference Example, please refer to Irrelevant Examples (if any) for
    examples of NL queries that are not relevant to the Customer use-case

## Reference Examples
- show the distance of the flights that arrived before last May
- visits made past more than twelve days
- show a list containing staff names and their respective genders who were assigned 2 days ago
- Find the names of the university which has more faculties in 2002 than every university in
    Orange county.
- What is all the information about employees hired until June 21, 2002?

## Irrelevant Examples
- show oldest ship in the port of Singapore

## Input Schema
CREATE TABLE ports(
id INT,
name VARCHAR(255),
country VARCHAR(255)
)

CREATE TABLE cargoes(
id INT,
name VARCHAR(255),
tonnage INT,
port_id INT,
load_date DATE
)

## Generated Examples
```

Figure 7: Prompt Example for NL Synthesizer Pipeline (AddRef).

Prompt Example for SQL Synthesizer Pipeline (LearnPrior)

```
Here is the database schema context:

CREATE TABLE ports(
    id INT,
    name VARCHAR(255),
    country VARCHAR(255)
)

CREATE TABLE cargoes(
    id INT,
    name VARCHAR(255),
    tonnage INT,
    port_id INT,
    load_date DATE
)

DateTime Instructions:
- With a DATE_column, refer to the following instructions:
   - 'today': TRUNC(DATE_column) = TRUNC(SYSDATE)
   - 'yesterday': TRUNC(DATE_column) = TRUNC(SYSDATE)-1
   - 'tomorrow': TRUNC(DATE_column) = TRUNC(SYSDATE)+1
   - 'this year': EXTRACT(YEAR FROM DATE_column) = EXTRACT(YEAR FROM SYSDATE)
   - 'this month': TO_CHAR(DATE_column, 'YYYY-MM') = TO_CHAR(SYSDATE, 'YYYY-MM')
   - 'last month': TO_CHAR(DATE_column, 'YYYY-MM') = TO_CHAR(ADD_MONTHS(SYSDATE, -1)
       'YYYY-MM')
   - 'next month': TO_CHAR(DATE_column, 'YYYY-MM') = TO_CHAR(ADD_MONTHS(SYSDATE, +1)
       'YYYY-MM')
   - 'until last month' TO_CHAR(DATE_column, 'YYYY-MM') <= TO_CHAR(ADD_MONTHS(SYSDATE, -1)
       'YYYY-MM')
   - 'until next month' TO_CHAR(DATE_column, 'YYYY-MM') <= TO_CHAR(ADD_MONTHS(SYSDATE, +1)
       'YYYY-MM')
   - 'this quarter': TRUNC(DATE_column, 'Q') = TRUNC(SYSDATE, 'Q')
   - 'last quarter': TRUNC(DATE_column, 'Q') = ADD_MONTHS(TRUNC(SYSDATE, 'Q'), -3)
   - 'last year': TO_CHAR(DATE_column, 'YYYY') = TO_CHAR(ADD_MONTHS(SYSDATE, -12), 'YYYY')
   - 'in the last X months': DATE_column BETWEEN ADD_MONTHS(SYSDATE, -X) AND SYSDATE
   - 'in the last X quarters': DATE_column ADD_MONTHS(TRUNC(SYSDATE, 'Q'), -3*X) AND
       TRUNC(SYSDATE, 'Q')
   - 'in the last X years': DATE_column BETWEEN ADD_MONTHS(SYSDATE, -12*X) AND SYSDATE
   - 'in next X days': (TRUNC(DATE_column) BETWEEN TRUNC(SYSDATE) AND TRUNC(SYSDATE) + X)
   - 'in year XXXX': EXTRACT(YEAR FROM DATE_column) = XXXX
   - 'after year XXXX': EXTRACT(YEAR FROM DATE_column) > XXXX
   - 'day X of month Y of year Z': TO_CHAR(DATE_column, 'YYYY-MM-DD') = 'ZZZZ-MM-XX'
   - 'after day X of month Y of year Z': DATE_column > TO_DATE( 'ZZZZ-YY-XX', 'YYYY-MM-DD')
   - 'next week': TO_CHAR(dueDate, 'YYYY-IW') = TO_CHAR(SYSDATE + 7, 'YYYY-IW')
   - 'in this February: EXTRACT(YEAR FROM DATE_column) = EXTRACT(YEAR FROM SYSDATE) AND
       EXTRACT(MONTH FROM DATE_column) = 2
   - 'in this October: EXTRACT(YEAR FROM DATE_column) = EXTRACT(YEAR FROM SYSDATE) AND
       EXTRACT(MONTH FROM DATE_column) = 10
   - 'in last February': EXTRACT(YEAR FROM DATE_column) = EXTRACT(YEAR FROM SYSDATE) - 1 AND
       EXTRACT(MONTH FROM DATE_column) = 2
   - 'in next February': EXTRACT(YEAR FROM DATE_column) = EXTRACT(YEAR FROM SYSDATE) + 1 AND
       EXTRACT(MONTH FROM DATE_column) = 2
   - 'from this April': TRUNC(DATE_column, 'MM') >= ADD_MONTHS(TRUNC(SYSDATE, 'YYYY'), 4-1) #
       beginning of this year + 3 months to align with start of April (EXTRACT(MONTH not
       needed here)
   - 'from this January': TRUNC(DATE_column, 'MM') >= TRUNC(SYSDATE, 'YYYY') # beginning of
       this year + 0 months to align with start of January (EXTRACT(MONTH not needed here)
   - 'from this October': TRUNC(DATE_column, 'MM') >= ADD_MONTHS(TRUNC(SYSDATE, 'YYYY'),
       10-1) # beginning of this year + 9 months to align with start of October
       (EXTRACT(MONTH not needed here)
   - 'until this February': TRUNC(DATE_column, 'MM') <= ADD_MONTHS(TRUNC(SYSDATE, 'YYYY'),
       2-1) # beginning of this year + 1 months to align with start of February
       (EXTRACT(MONTH not needed here)
   ... (truncated)

Question: What's the total tonnage of all cargoes loaded or unloaded at the port of Singapore
    before last April

Write a SQL query in Oracle SQL dialect, compatible with the latest version of Oracle
    Database, that answers the question above.
```

Figure 8: Prompt Example for SQL Synthesizer Pipeline (LearnPrior).

LLMs-as-Juries Quality Evaluation Prompt Example

```
Given an input Question and a Oracle SQL query, prepare an assessment based on the following
     criteria:

SQL Correctness
    - Add one star if the Oracle SQL query returns incorrect results
    - Add one more star, i.e. award 2 stars if the Oracle SQL query executes but returns
        partially correct results
    - Add one more star, i.e. award 2 stars if the Oracle SQL query returns mostly correct
        results but with minor inaccuracies or omissions
    - Add one more star, i.e. award 2 stars if the Oracle SQL query returns correct results
        with negligible issues
    - Add one more star, i.e. award 2 stars if the Oracle SQL query returns accurate and
        complete results as per the requirement

Compliance with Oracle SQL Standards
    - Add one star if the SQL query does not follow Oracle SQL standards or best practices,
        using deprecated or non-standard syntax
    - Add one more star, i.e. award 2 stars if the SQL query loosely follows Oracle SQL
        standards, with several deviations from best practices.
    - Add one more star, i.e. award 2 stars if the SQL query generally follows Oracle SQL
        standards but has room for better alignment with best practices.
    - Add one more star, i.e. award 2 stars if the SQL query closely follows Oracle SQL
        standards and adheres to many best practices.
    - Add one more star, i.e. award 2 stars if the SQL query strictly adheres to Oracle SQL
        standards and best practices, showcasing exemplary coding standards.

Quality of the Natural Language Query
    - Add one star if the natural language query does not match the SQL, or cannot be
        answered given the provided Schema.
    - Add one more star, i.e. award 2 stars if the natural language query matches the SQL,
        but the question does not make any sense to be asked (totally unrealistic).
    - Add one more star, i.e. award 3 stars if the natural language query is consistent with
        the SQL, but it it does not look natural (no domain knowledge, the style looks
        synthetic-templated, does not use "domain-specific" words).
    - Add one more star, i.e. award 4 stars if the natural language query is correct and
        consistent, but the NL Question can further be improved for clarity, conciseness,
        small typos.
    - Add one more star, i.e. award % stars if the natural language query is perfect.

The Schema context is provided below.

CREATE TABLE ports(
    id INT,
    name VARCHAR(255),
    country VARCHAR(255)
)

CREATE TABLE cargoes(
    id INT,
    name VARCHAR(255),
    tonnage INT,
    port_id INT,
    load_date DATE
)

Question: What's the total tonnage of all cargoes loaded or unloaded at the port of Singapore
     before last April
Oracle SQL: SELECT SUM(c.tonnage) FROM cargoes c JOIN ports p ON c.port_id = p.id WHERE
     p.name = 'Singapore' AND (EXTRACT(YEAR FROM c.load_date) < EXTRACT(YEAR FROM SYSDATE) -
     1 OR (EXTRACT(YEAR FROM c.load_date) EXTRACT(YEAR FROM SYSDATE) - 1 AND EXTRACT(MONTH
     FROM c.load_date) < 4));

The output must have following items in an orderly manner:
- The final star ratings of criterions in a list-wise manner
- The final star ratings of criterions in a json format
- Explain the scores with a short text (< 100 words).
```

Figure 9: Prompt for LLM-based Quality Evaluation.

LLMs-as-Juries Relevance Evaluation Prompt Example

```
Given an a Natural Language query and the corresponding SQL Query generated for a NL2SQL
    Model, your goal is to assess whether the generated example is relevant to the Customer
    use-case represented by any of the Reference Examples shown below.

## Reference Examples
- show the distance of the flights that arrived before last May
- visits made past more than twelve days
- show a list containing staff names and their respective genders who were assigned 2 days ago
- Find the names of the university which has more faculties in 2002 than every university in
    Orange county.
- What is all the information about employees hired until June 21, 2002?
- Show me the aircraft names that travelled 8430 kms that departed before November of 4 years
    ago
- How many students exist who are registered with just a single allergy?
- show all maintenance contracts that end until next Dec
- Give me the list of actors which was last updated until last Saturday
- show the distance of the flights that arrived before last January
- show all machines made in 1992
- Show me invoices that are due to be paid in the next half year.
- What is all the information about employees hired until June 21, 2002?
- show all order items delivered before last march
- Show the number of attendees in year 2008 or 2010.
- Show me all students who registered for a course from 3 days ago, including the course name
    and student details.
- give people addresses who lived on address till april.
- List all customers who placed an order from the next 30 days and the order status is 'New'.
- show all maintenance contracts that end until next May
- How many customers are not responded to mailshot sent from week 5 2018

## Input Natural Language query and SQL query
Natural language query: What's the total tonnage of all cargoes loaded or unloaded at the
    port of Singapore before last April
SQL Query: SELECT SUM(c.tonnage) FROM cargoes c JOIN ports p ON c.port_id = p.id WHERE p.name
    = 'Singapore' AND (EXTRACT(YEAR FROM c.load_date) < EXTRACT(YEAR FROM SYSDATE) - 1 OR
    (EXTRACT(YEAR FROM c.load_date) EXTRACT(YEAR FROM SYSDATE) - 1 AND EXTRACT(MONTH FROM
    c.load_date) < 4));

## Assessment ("**Relevant**"/"**Irrelevant**")
```

Figure 10: Prompt for LLM-based Relevance Evaluation.