Unlocking SLM Potential for Data Analysis Code Generation via Non-Parametric Knowledge Distillation

Jinyang Li[α,β]*, **Jack Williams**[α], **Nick McKenna**[α], **Arian Askari**[α], **Nicholas Wilson**[α], **Reynold Cheng**[β]
[α]Microsoft Research Cambridge [β]The University of Hong Kong
jl0725@connect.hku.hk, jack.williams@microsoft.com, ckcheng@cs.hku.hk

Abstract

Knowledge distillation from Large Language Models (LLMs) to locally hosted Small Language Models (SLMs) provides advantages for Data Analysis Code Generation (DACG) such as privacy protection. However, achieving effective distillation without resource-intensive training is challenging. This paper investigates whether LLMs can distill knowledge to SLMs through In-Context Learning (ICL), a training-free method for rapid task adaptation. We present the **DARGO**: ***D**istillation and **A**daptive **R**easoning-**G**uided **O**rchestration* framework, which facilitates automatic knowledge distillation from LLMs to SLMs. DARGO consists of three phases: exploration through an **Model Orchestration Interface (MOI)**, **Memory Collection** of successful trajectories, and **Knoweldge-driven Inference**. We evaluate DARGO on three challenging DACG benchmarks (WIKITQ, TABMWP, and BIRD-SQL), each with in-domain training sets that enable detailed analysis of knowledge distillation effectiveness. DARGO demonstrates a substantial relative performance improvement of 27.5% on average for the student SLMs. To further observe generalization capabilities, we evaluate the DARGO across different teacher-student model combinations, knowledge transfer scenarios, and unified memory approaches for more advanced, test-only data analysis tasks. Our findings contribute a novel perspective on distillation methods that enhance performance for SLMs while avoiding intensive fine-tuning. The source code is available: https://github.com/accpatrick/DarGO.

1 Introduction

Data Analysis Code Generation (DACG) automates the conversion of natural language queries into executable code, empowering information extraction and analysis from tabular data efficiently. This process enhances productivity, reduces the technical barrier for data analysis, and allows data scientists to focus on deriving insights, ultimately supporting more effective decision-making [24, 17, 14]. This is a challenging task since it not only requires the capability of code generation but also understanding complex tabular data.

Large Language Models (LLMs) have demonstrated remarkable performance across diverse, complex tasks [49, 37, 8, 71, 11]. Leveraging LLMs or LLM agents for automatic code generation from user queries offers an effective solution [65, 56]. However, the integration of LLMs in DACG faces two primary challenges: 1) Privacy concerns arise when utilizing closed-source LLMs such as GPT-4 [2] or Claude-3.5-Sonnet [40]. 2) Deploying large open-source models like Llama-3.1-405B [13] or DeepSeek-v3 (671B) [31] can be challenging due to their large number of parameters. Balancing these benefits and challenges is crucial for effective data science applications.

*Work done during Research Intern at Microsoft Research Cambridge

39th Conference on Neural Information Processing Systems (NeurIPS 2025).

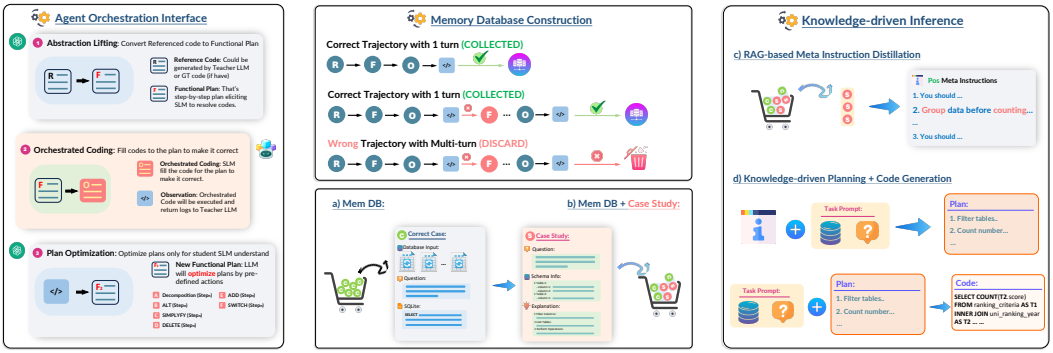


Figure 1: Overview of the orchestration system of DARGO for data science code generation. Left: Model Orchestration Interface (MOI) with abstraction lifting, orchestrated coding, and plan optimization. Center: Memory Database Construction, including trajectory collection and case study integration. Right: Knowledge-driven Inference and Planning, featuring RAG-based meta instruction generation, and knowledge-driven code generation. **R:** Referenced GT Code. **F:** Functional Plans. **O:** Orchestrated Coding.

Small Language Models (SLMs), such as Phi-3-mini [1] and Llama-3.1-8B [13], have gained attention for their In-Context Learning (ICL) capabilities and ability to be locally deployed. These models offer computational efficiency and enhanced data privacy, crucial for resource-constrained or privacy-sensitive applications [21]. While SLMs have shown competitive performance in some general tasks including natural language understanding [39] and code completion [7], their effectiveness in data science code generation tasks remains an open question.

Fine-tuning is a common strategy to enhance SLM capabilities for complex tasks [42]. However, this approach encounters several challenges in the domain of data science DACG. One primary issue is the limited availability of high-quality training data. Professional tabular datasets, such as relational databases, are often small or proprietary, restricting access to substantial corpora for training. Additionally, the dual expertise required in both coding syntax and data understanding for accurate annotation further constrains dataset scalability [30, 28]. This is reflected in recent benchmarks for data science code generation, which typically contain around or fewer than 1,000 samples, highlighting the complexity and resource constraints in this field [20, 3, 26, 70, 66]. Recent research has explored distillation from LLMs to SLMs through fine-tuning on synthetic data [52, 36, 22]. While this approach shows promise, several challenges persist. For example, the performance improvements obtained from fine-tuning approaches can fail to generalize across different programming languages or dialects, requiring re-training for each package update or new task [46, 23]. However, In-Context Learning (ICL) can adapt to new requirements or tasks by providing relevant instructions or examples, reducing the effort required for re-training or continual training. This raises our central research question in DS code generation: ***Can LLMs distill knowledge to SLMs through In-Context Learning (ICL)?***

In this paper, we explore the potential of knowledge distillation from LLMs to SLMs via ICL. We propose a novel **D**istillation and **A**daptive **R**easoning-**G**uided **O**rchestration framework that enables an LLM to serve as a Teacher model guiding SLMs (Student models) in complex DACG tasks. DARGO operates in three phases: **exploration**, **memory database collection**, and **knowledge-driven inference**. During exploration, we employ the Model Orchestration Interface (**MOI**) that allows an LLM to probe and analyze SLM code knowledge by converting questions into step-wise functional plans and asking SLMs to infill the code for each plan. Then, successful collaborated cases are stored in a memory databases. In the knowledge-driven inference phase, DARGO utilizes a retrieval-augmented generation (RAG) approach that dynamically distills knowledge at inference time by presenting relevant prior successful cases in a case-study format to guide the generation process, which proves friendly to SLMs to absorb (Section 2.3).

We evaluate DARGO on three challenging tabular analysis benchmarks requring code generation:TABMWP [34], BIRD-SQL [29], and WIKITQ [41], where each feature fixed, non-overlapping train–test splits to closer show how effective of knowledge distillation that DARGO is. The experimental results demonstrate that the DARGO framework significantly improves the performance

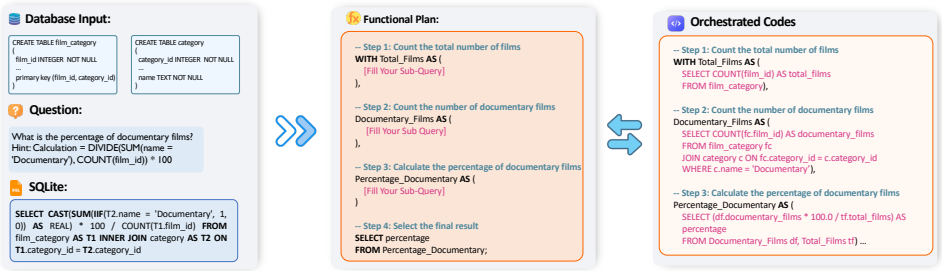


Figure 2: Main steps of MOI demonstrated with a text-to-SQL task example. Teacher model converts referenced code to a functional plan for Student models to complete. Teacher model iteratively optimizes the plan until the Student model produces correct code. A Python code MOI example is provided in Appendix D.1.

of SLMs across all datasets, validating the potential of our knowledge distillation approach via ICL. Importantly, the memory produced for one student can guide *other* students, demonstrating model-agnostic transfer. Additional cross-dataset experiments on CRT-QA [70], QRDATA [32], and Infi-Agent [20] confirm that the distilled knowledge generalizes beyond the original training distribution. Taken together, our results show that lightweight, privacy-preserving SLMs can inherit much of an LLM's analytic expertise through ICL alone, making DACG practical in resource-constrained or privacy-sensitive environments.

2 Methodology

2.1 Task Formulation

Given a natural language query $q_i \in \mathcal{Q}$, where $\mathcal{Q} = \{q_1, q_2, \ldots, q_N\}$ represents a set of N queries, the corresponding tabular data or database schema information $d_i \in \mathcal{D}$, where $\mathcal{D} = \{d_1, d_2, \ldots, d_N\}$, then the Small Language Model (SLM) is tasked with generating an executable code snippet c_i. This code snippet must accurately answer the query q_i with the associated data d_i. The function that maps each query-data pair to its corresponding code snippet by SLMs is defined as f_{gen}, and can be written as:

$$c_i = f_{\text{gen}}(d_i, q_i) \quad \text{for } i = 1, 2, \ldots, N. \tag{1}$$

2.2 Model Orchestration Interface

The Model Orchestration Interface (MOI) is conducted between a Large Language Model (LLM), a Teacher model, and a Student model, represented by the SLM. A Teacher LLM with superior reasoning capabilities will often generate plans that are too abstract for a weaker Student SLM to enact. These plans require decomposition and refinement to match the operational granularity of the Student SLM. Through orchestrated mediation, the MOI dynamically adjusts plan granularity to align with the SLM's code capabilities whilst maintaining task conformance. The MOI is composed of three key components: Abstraction Lifting, Orchestrated Coding, and Plan Optimization.

Abstraction Lifting (AL). In this phase, LLM generates a functional plan $P_i = \{s_{i1}, s_{i2}, \ldots, s_{iK}\}$ based on a query q_i, data input d_i, and the corresponding ground truth (gt) code $\tilde{c}_i$. The ground truth code $\tilde{c}_i$ can either be sourced from an existing dataset (BIRD-SQL) or generated by the Teacher model when it is not directly available but a ground-truth answer string exists, such as WIKITQ and TABMWP. This functional plan is defined as $\mathcal{L}_{\text{al}}(d_i, q_i, \tilde{c}_i)$, where $\mathcal{L}_{\text{al}}$ denotes the LLM performing abstraction lifting. Each step s_{ij} in the plan corresponds to a key subtask derived from the query, collectively forming a structured template outlining the solution process. These steps are annotated by the LLM with descriptive comments and placeholders such as `[Fill Your Code Here]` in Python or `[Fill Your Sub-Query]` in SQL, as shown in Figure 2, ensuring that the SLM follows the logical flow of the entire plan and enables guided code generation. Unlike Chain-Of-Thought [60] plans, which provide intermediate steps in continuous textual form, our approach

Table 1: The 6 action types utilized by the LLM during the Model Orchestration Interface (MOI) to optimize the plans for better understanding and code generation by SLMs

Action Type	Expression	Description
Decomposition	`step(x)` → `step(a), step(b)`	Split a complex step `x` into smaller, manageable steps such as step `a` and step `b`.
ALT	`step(x)` → `step(y)`	Replace a step `x` described by ambiguous or incorrect messages with a clearer and correct alternative step `y`.
ADD	`step(x)` → `step(x), step(a)`	Add a necessary step `a` to ensure the completeness of code logic.
DELETE	`step(x)` → `None`	Remove the unnecessary step `x`, which may lead to misunderstanding by the SLM.
SIMPLIFY	`step(x)` → `simple_step(x)`	Replace a complex step `x` with a simpler approach. For example, convert recursive plans into iterative loops.
SWITCH	`packageA.step(x)` → `packageB.step(x)`	Use a simpler package to achieve the same functionality. For example, conversion from Package `Linear Regression` to `Correlation Coefficient` to determine relationship between two variables.

bridges high-level problem understanding with low-level code implementation logic, allowing the SLM to more effectively follow the plan for DACG.

Orchestrated Coding (OC). Once the functional plan P_i is provided, the Student SLM considers all context including the question and data input to generate the complete orchestrated code $c_i = f_{\text{gen}}(d_i, q_i, P_i)$ in a single turn by filling all placeholders, ensuring the solution is correct and executable. The results from executing this orchestrated code are then compared to those from a reference solution (such as ground truth answer string or gt codes) to evaluate whether the SLM fully understands both the data and the logic needed to answer the question. This comparison serves as a key indicator of the problem-solving accuracy of SLM and alignment with the intended solution. While the ground truth code may already be available from datasets or generated by the Teacher model, orchestrated coding and abstraction lifting are crucial for a few reasons. **First**, AL breaks down complex problem-solving tasks into manageable sub-tasks, with the potential to improve the performance of the SLMs across a wide range of analytical queries by assisting them in understanding modular structure. **Additionally**, error isolation can be grounded in the program structure, enabling more precise identification of issues and contributing to optimized plans. This is supported by our analysis in Section 3.6 that compares chain-of-thought with functional plans for exploration.

Plan Optimization (PO). The plan optimization process is an iterative procedure that unfolds over multiple turns, denoted by t. During each iteration, the SLM refines the functional plan P_i^t. To formalize this interactive optimization process, we define an environment $\mathcal{E} = \langle \mathcal{S}, \mathcal{A}, \mathcal{O}, \mathcal{T} \rangle$, following [72, 62, 16], where $\mathcal{S}$ represents the state space, $\mathcal{A}$ the action space (Table 1), and $\mathcal{O}$ the observation space. In this context, the plan P_i^t is embedded within the current state $\mathcal{S}_i^t$, serving as a structure that guides the SLM to generate code. The orchestrated code c_i^t is the snippet produced by performing the plan P_i^t within the environment.

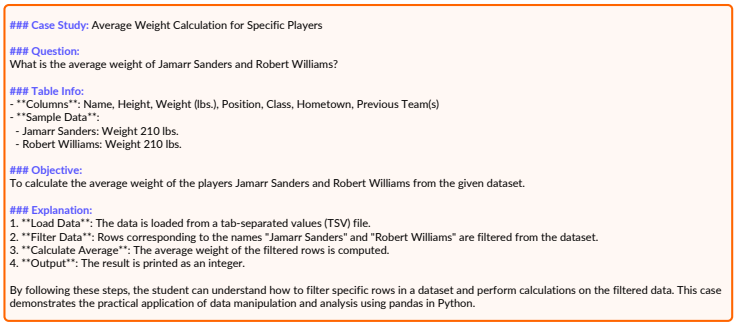
During each turn, the LLM observes o_i^t, the outcome of executing orchestrated code c_i^t generated by the SLM, and selects an action a_i^t from $\mathcal{A}$ to optimize the plan. For example, if a step s_{ij}^t contains an error such as: "`Step j: List players who was born before 1930 and after 1950`", the LLM will apply an action `ALT(·)` to correct this, resulting in an updated plan P_i^{t+1} with the refined step s_{ij}^{t+1} as "`Step j: List players who were born after 1930 and before 1950`". This iterative process can be represented as:

$$
\begin{aligned}
P_i^{t+1} &= \{s_{i1}^t, s_{i2}^t, \ldots, s_{ij}^{t+1}, \ldots, s_{iK}^t\}, \\
s_{ij}^{t+1} &= \mathcal{L}_{\text{opt}}\big(s_{ij}^t,\, o_i^t,\, a_i^t\big).
\end{aligned}
\tag{2}
$$

Here, s_{ij}^{t+1} is updated by the optimization function $\mathcal{L}_{\text{opt}}$ of the LLM, which integrates the current observation o_i^t, action a_i^t and sub-optimal step s_{ij}^t. The system transitions from state $\mathcal{S}_i^t$ to $\mathcal{S}_i^{t+1}$ through $\mathcal{T}(\mathcal{S}_i^t, \mathcal{A}_i^t)$, resulting in the updated plan P_i^{t+1}. The SLM then generates the updated orchestrated code $c_i^{t+1} = f_{\text{gen}}(q_i, d_i, P_i^{t+1})$ for the new plan. This process repeats until the output is correct or the maximum number of iterations T is reached.

2.3 Memory Database Construction

After interactions between the LLM and SLM in MOI, the finalized states are stored in a memory database. This database includes the correct orchestrated codes, along with the context of the question and data input. This process ensures that the SLM can efficiently reference and apply related knowledge to new, unseen queries.



Case Study: Average Weight Calculation for Specific Players

Question:
What is the average weight of Jamarr Sanders and Robert Williams?

Table Info:
- **Columns**: Name, Height, Weight (lbs.), Position, Class, Hometown, Previous Team(s)
- **Sample Data**:
  - Jamarr Sanders: Weight 210 lbs.
  - Robert Williams: Weight 210 lbs.

Objective:
To calculate the average weight of the players Jamarr Sanders and Robert Williams from the given dataset.

Explanation:
1. **Load Data**: The data is loaded from a tab-separated values (TSV) file.
2. **Filter Data**: Rows corresponding to the names "Jamarr Sanders" and "Robert Williams" are filtered from the dataset.
3. **Calculate Average**: The average weight of the filtered rows is computed.
4. **Output**: The result is printed as an integer.

By following these steps, the student can understand how to filter specific rows in a dataset and perform calculations on the filtered data. This case demonstrates the practical application of data manipulation and analysis using pandas in Python.

Figure 3: An example of a generated case study to enhance comprehension for SLMs.

Case Study Translation. Rather than only storing raw, heterogeneous cases that consist of a query, plan, and orchestrated code in a simple stacked format, the LLM refines these into case study-like representations. These representations distill the reasoning behind the success of each example, serving as an intermediate abstraction that emphasizes the underlying rationale for the chosen approach. Each case study G_i contains a `Case Name`, `Question`, `Schema / Value Information`, `Objective`, and an `Explanation` of how the solution code successfully addresses the query using the provided data. An example of this structure is provided in Figure 3. As shown in Figure 1 (a)-(b), DARGO performs case study translation only for correct cases, because reflecting on incorrect cases without supervision often introduces hallucinations.

Correct Case Collection. The Correct Case Collection, denoted as $\mathcal{M}$, consists of cases where the SLM has generated correct orchestrated codes. Each case M_i in this collection contains the natural language query q_i, the corresponding data d_i, the correct orchestrated code $\hat{c}_i$, which contains descriptive comments as shown in Figure 2 (right), and the case study G_i illustrating the solution. The set $\mathcal{M}$ is the union of all such individual cases:

$$\mathcal{M} = \bigcup_i M_i, \quad M_i = (q_i, d_i, \hat{c}_i, G_i)\,. \tag{3}$$

2.4 Knowledge Distillation from Memory Database

This part presents a RAG-based method of knowledge distillation as a global instruction, termed as meta-instruction. This instruction then guides the SLM in learning how to plan and generate code more accurately for unseen queries.

RAG-based Knowledge Distillation for Meta Instruction Generation. We propose a Retrieval-Augmented Generation (RAG) framework for localized instruction distillation. In the retrieval phase, we identify the top relevant examples from a memory database $\mathcal{M}$ via an embedding model. Relevance is measured via a function $\mathcal{D}$, expressed as $\mathcal{R}(q_i) = \mathcal{D}(q_i, \mathcal{M}, k)$, where k is number of most relevant cases.

Model	WIKITQ	TABMWP			BIRD-SQL			
	Accuracy	Grad. 1-6	Grad. 7-8	Total	Sim.	Med.	Chal.	Total
CodeLlama-7B	11.80	26.55	13.11	20.50	43.92	18.00	11.76	24.40
CodeLlama-13B	34.90	37.27	24.22	31.40	45.27	19.60	17.65	26.80
StarCoder2-7B	20.70	34.00	27.56	31.10	41.22	21.60	17.65	26.60
StarCoder2-15B	36.60	39.09	33.14	36.50	43.92	29.20	14.71	30.60
Phi-3-Small-7B	27.00	46.36	38.00	42.60	52.03	28.40	10.78	31.80
Phi-3-Medium-14B	44.80	59.45	46.00	53.40	51.35	32.80	13.73	34.40
SLM Performance								
Phi-3-Mini-3.8B	32.50	44.18	38.89	41.80	38.51	21.20	11.76	24.40
+ Chain-Of-Thought	27.70	46.36	35.33	41.40	34.46	22.00	12.75	23.80
+ Static Few-Shot	23.00	37.27	34.89	36.20	47.97	20.80	7.84	26.20
+ Dynamic Few-Shot	16.60	51.45	52.89	52.10	42.57	18.80	11.76	24.40
+ DSPy Distillation (Few Shot)	26.70	42.70	37.60	40.40	36.00	19.80	11.00	22.80
+ ReGAL Distillation	36.10	44.73	38.22	41.80	35.14	14.80	12.75	20.40
+ DARGO Meta Instruction (Ours)	**41.10**	**61.27**	**56.44**	**59.10**†	**51.35**	**30.40**	**16.67**	**33.80**

Table 2: Performance comparison of various SLMs on WIKITQ, TABMWP, and BIRD-SQL, with results presented in accuracy percentages. Improvements of our DARGO methods over the End-to-End Code Gen baseline are highlighted using different intensities of olive color. **Bold** indicates best results for Phi-3-Mini, while underlines denote second-best results.[†] means a hybrid with dynamic few-shot with DarGO. Detailed instructions of when to use few shot is in Appendix A.4.

Then, as shown in Figure 1 (d), the case studies of these retrieved examples are then fed into the SLM to extrapolate plans for solutions, adhering them to the specific query. Here, SLM performs a secondary distillation, extracting shared knowledge patterns from these case studies, which have already been distilled by the LLM (Teacher model), to generate instructions, noted as **Meta-Instruction** ($\mathcal{I}_{\mathbf{m}}(q_i)$), precisely specific to the current query at hand. The process is formalized:

$$\mathcal{I}_{\text{m}}(q_i) = f_{\text{agg}}(q_i, \mathcal{R}(q_i)) \tag{4}$$

where f_{agg} is an aggregation function applied by the SLM. By doing so, SLMs can generate more relevant and contextually appropriate instructions, effectively bridging the gap between general knowledge and query-specific requirements.

Knowledge-Driven Inference. Harnessing the distilled instructions $\mathcal{I} = \mathcal{I}\text{m}(q_i)$ from the memory database, the SLM initially formulates a structured plan p_{gen}, which it subsequently employs to generate code for new queries (Figure 1 (e)). For a given query q_i and its associated data d_i, this process unfolds as follows:

$$P_{\text{gen}} = f_{\text{plan}}(\mathcal{I}, d_i, q_i), \quad c_i = f_{\text{gen}}(d_i, q_i, P_{\text{gen}}), \tag{5}$$

where f_{plan} denotes the planning function executed by the SLM. This plan serves as a blueprint, guiding the following code generation phase. The SLM then employs the function f_{gen}, which takes P_{gen} along with the original query q_i and data d_i to generate the final code c_i.

3 Experiments

In this section, we first describe datasets and evaluation metrics in Section 3.1, followed by implementation details in Section 3.2. We then present a comprehensive experiments aimed at addressing three key research questions:

- **RQ1 (Section 3.3)**: How effective is DARGO for DACG?
- **RQ2 (Section 3.4)**: Does DARGO and the knowledge it distills generalize across models?
- **RQ3 (Section 3.5)**: How does DARGO compare to popular Lora fine-tuning?
- **RQ4 (Section 3.6)**: Are all components of DARGO necessary?

3.1 Datasets and Metrics

We evaluate our approach on the DACG datasets WIKITQ [41], TABMWP [34], and BIRD-SQL [29]. These datasets vary in data complexity and task requirements. Full dataset statistics are provided in Appendix B.1. WIKITQ features operations such as counting, comparison, and aggregation, and we

Table 3: Zero-shot performance of DARGO on different SLMs.Improvements (in parentheses) show gains over the Baseline.

Model	WIKITQ	TABMWP	BIRD-SQL
StarCoder-15B as Student			
Baseline	36.60	36.50	30.60
+ DARGO-MI	45.70 (↑9.10)	43.93 (↑7.43)	38.40 (↑7.80)
Llama-3.1-8B as Student			
Baseline	34.30	42.90	40.20
+ DARGO-MI	39.80 (↑5.50)	49.10 (↑6.20)	44.20 (↑4.00)

Table 4: Zero-shot performance of DARGO on different teacher LLMs. → means distill knowledge from teacher LLMs to student SLMs.

Model	WIKITQ	TABMWP	BIRD-SQL
Llama-3.3-70B → Phi-3-mini			
Baseline	41.80	32.50	24.40
+ DARGO-MI	48.80 (↑7.00)	42.60 (↑10.10)	36.20 (↑11.80)
Llama-3.3-70B → Llama-3.1-8B			
Baseline	42.90	34.30	40.20
+ DARGO-MI	50.40 (↑7.50)	42.00 (↑7.70)	47.00 (↑6.80)

select 1,000 test and 2,000 training examples. Accuracy is measured using the official scripts from Pasupat and Liang [41]. TABMWP involves mathematical word problems in tabular data, with 1,000 questions used for memory construction and a separate 1,000 test set. Performance is evaluated by comparing answers to ground truth. BIRD-SQL presents relational databases with both semantic parsing and analytical tasks. We adopt the 1,000-example mini-train set from Qu et al. [44] and evaluate on a 500-example mini-dev set using execution accuracy (EX).

3.2 Implementations

Setup. We conduct experiments in two settings. For TABMWP and WIKITQ, we follow Stengel-Eskin et al. [50] to first have a Teacher LLM (GPT-4o) generate silver programs during exploration. At inference, an SLM produces Python code, which is executed to obtain a final answer string for comparison with the ground truth answer. For BIRD-SQL, no silver code is required because the dataset already contains ground-truth SQL. RAG-based Meta Instructions employ $k = 3$ nearest neighbors (via CodeT5+ [59]). Further implementation details appear in Appendix B.

Baselines Models. We consider an SLM suitable if it can perform in-context learning and has under 15B parameters for one A100 80G GPU inference. For closed-source models, we choose GPT-35-Turbo as SLM since it has faster inference speed and its performance falls behind advanced models (e.g., GPT-4). We classify them into three categories: *Orchestration Models*, where GPT-4o is the Teacher, and Phi-3-mini-128k [1], Llama-3.1-8B [13] serve as Students for verifying the generalization of our approach. *Evaluation Models*, including Phi-3, CodeLlama, and StarCoder2 families. *Knowledge-Transmission Models*, containing GPT-35-Turbo and Llama-3.1-8B, used to test knowledge transmission in Section 3.4.

Baseline Methods Implementation. Baseline methods include zero-shot end-to-end, Chain-of-Thought [60], static few-shot [6], and dynamic RAG-based few-shot [15] with memory database by DARGO since python tasks do not have GT codes but answer string, each with three examples. We also compare two distillation frameworks, DSPy [25] and ReGAL [51], both using GPT-4o as the teacher.

3.3 Overall Results

Table 2 highlights three key aspects:

1) Effectiveness of DARGO: Distilling knowledge via **DARGO** propels Phi-3-mini to outperform both End-to-End Code Generation and Chain-of-Thought on all SLM datasets, achieving relative gains of 17.5% on TABMWP to 38.5% on BIRD-SQL. Moreover, Phi-3-mini with DARGO often surpasses larger models (2–3× parameters), outperforming CodeLlama-13B by 7.0% and StarCoder2-15B by 3.2% on BIRD-SQL, and rivaling Phi-3-Medium (4× parameters) across all benchmarks.

2) Comparison with Advanced Distillation Strategies. We implement two advanced distillation pipelines, DSPY and REGAL, on GPT-4o and Phi-3-mini for fair comparison, reporting the best performance of REGAL across different numbers of reusable helper functions (1–10) to minimize bias. As shown in Table 2, **DARGO** outperforms both approaches on all three datasets and difficulty levels.

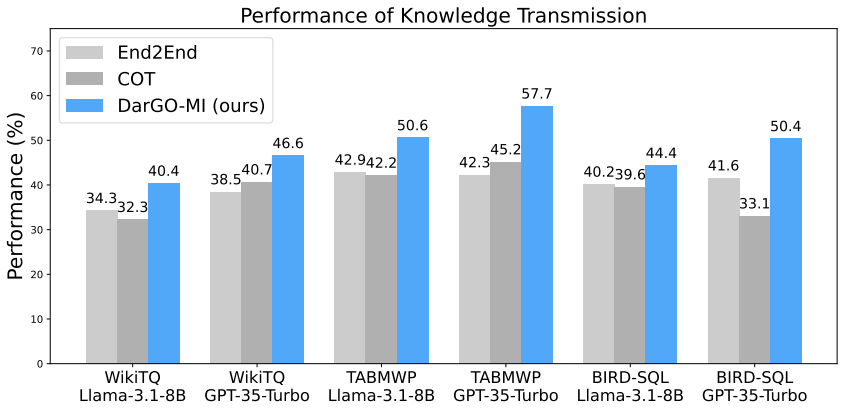


Figure 4: Knowledge transmission for from the memory database between GPT-4o and Phi-3-mini across three datasets.

DSPy is robust for general QA but struggles in execution-based DACG settings because its exact-match metric does not account for valid alternative program solutions. Moreover, DSPy relies on few-shot demonstrations rather than comprehensive knowledge distillation, reflecting the broader challenges of demonstration-based methods for complex DACG. By contrast, **DARGO** prioritizes task-specific instructions to guide complex operations, shifting focus from prompt optimization to experience summarization.

We also evaluate REGAL, which provides Reusable Python Helper Functions from the memory database. Although promising for tightly scoped, domain-specific tasks, REGAL proves less effective for cross-domain DACG (see Table 2). Error analysis shows it often produces narrowly tailored functions (e.g., `get_least_points_team`, `filter_senator_by_year`), which often directly reference the data schema and are therefore irrelevant on new tasks. In contrast, the distilled knowledge of DARGO is textual, focusing on reasoning-level challenges, and is therefore not syntactically fixed to a specific domain or input.

3.4 Generalization

Here, we conduct experiments to test generalization of DARGO framework:

Is DARGO model-agnostic to SLMs? To investigate the breadth of the DARGO pipeline beyond a single teacher–student configuration, we perform two complementary experiments. (1) We apply the complete workflow to two additional student models: `StarCoder2-15B` and `Llama-3.1-8B` while keeping `GPT-4o` as the teacher. As summarized in Table 3, both students outperform their respective baselines on all three datasets, indicating that DARGO consistently improves a variety of SLM architectures. (2) To examine robustness with respect to the teacher LLM, we replace `GPT-4o` with `Llama-3.3-70B`. The results in Table 4 replicate the earlier gains, demonstrating that gains of DARGO are not confined to a specific teacher model.

Is the distilled knowledge only useful to the student model participating in Orchestration? Next, we examine whether the knowledge distilled through DARGO is only beneficial to the student model involved in the orchestration phase. Even though this has been proven effective, it would be costly if exploration phase is repeated for new SLMs. Therefore, we test the distilled instructions on two new SLMs, `Llama-3.1-8B` and `GPT-35-Turbo`, both of which have not participated in the exploration between `GPT-4o` and `Phi-3-mini`.

Figure 4 shows that meta instructions (MI) derived from our memory database (jointly updated by GPT-4o and `Phi-3-mini`) produce siginificant improvements over baselines. Specifically, `Llama-3.1-8B` achieves an average relative improvement of 14.3%, while `GPT-35-Turbo` sees a 30.9% gain. These results indicate that the distilled knowledge is not tied to a specific student model; it can effectively transfer to new models without additional fine-tuning, offering a scalable means of augmenting emerging SLMs.

Table 5: Performance of DARGO on different SLMs. Improvements (in parentheses) are over the Baseline.

Model	CRTQA	QRDATA	INFI-AGENT
Phi-3-mini as Student			
Baseline	26.51	31.14	42.80
+ DARGO-MI	32.12 (↑5.61)	38.99 (↑7.85)	46.69 (↑3.89)
Llama-3.1-8B as Student			
Baseline	35.44	34.43	49.81
+ DARGO-MI	45.88 (↑10.44)	44.81 (↑10.38)	54.18 (↑4.37)

Table 6: Ablation study of exploration in DARGO on BIRD-SQL.

METHOD	BIRD-SQL (MI)
DARGO w/ Phi-3-mini	**33.80**
(a) w/o MOI	21.70 (↓12.10)
(b) w/o Case Study Trans.	22.80 (↓11.00)
(c) w/o Functional Plan	23.60 (↓10.20)

Out-of-Distribution Evaluation of DARGO Current tabular-reasoning benchmarks frequently provide *test-only* splits due to large costs of more complex tasks annotation, prompting us to evaluate how well DARGO generalizes when *no* in-domain training data are available. Following the [55], we first construct a unified memory from our exploration corpus and evaluate its performance on those *test-only* sets via DARGO. To be specific, exploration set $\mathcal{B}$ contains: WikiTQ [41], TabMWP [34], BIRD-Pandas [29], and Juice [3]. The distributions of $\mathcal{B}$ are shown in Appendix B.4. The *test-only* datasets form the set $\mathcal{E}$, which includes CRT-QA [70], QR-DATA [32], and Infi-Agent [20].

During evaluation, when we test DARGO on benchmark $\mathcal{E}_i \in \mathcal{E}$, its corresponding exploration corpus is $\mathcal{B} \cup \mathcal{E}_{\setminus i}$. This strict partition guarantees zero overlap between examples used for distillation and those reserved for evaluation. As summarized in Table 5, DARGO presents huge OOD improvements across all student models, demonstrating that it distills transferrable reasoning patterns rather than memorizing dataset-specific artifacts.

3.5 Fine-Tuning v.s. DARGO Knowledge Distillation

With the Same Seed Data. We compare DARGO with LoRA fine-tuning [19] using the same 1,000 training samples on BIRD-SQL for Phi-3-mini. Figure 5 and Table 9 in Appendix shows that fine-tuning on limited data even degrades SLM performance, while DARGO achieves significant improvements. We attribute this to two factors: 1) small training sets introduce bias that limits generalization; and 2) LoRA struggles to teach complex reasoning capabilities in an end-to-end regime. In contrast, DARGO leverages LLMs to decompose difficult questions into interpretable steps and distill planning knowledge, enabling better generalization. This makes DARGO particularly effective for DACG domains with limited annotations.

Comparison with Incremental Lora Fine-Tuning. For a more comprehensive comparison, we conduct additional experiments fine-tuning Phi-3-mini on varying proportions (10%, 20%, 50%, 70%, 100%) of the full BIRD-SQL training set. Each configuration is repeated 5 times, with average results presented in Figure 5. Our findings reveal several notable patterns: First, performance peaks at 70% of the data, suggesting that fine-tuning on more data does not always lead to better results, which maybe due to the inclusion of lower-quality or biased examples in the full dataset. Second, while fine-tuning with 70-100% of the data marginally outperforms DARGO, **it requires 7-9× more training examples** to achieve this performance. Notably, DARGO outperforms fine-tuning on 50% of the data, demonstrating **5× greater data efficiency**.

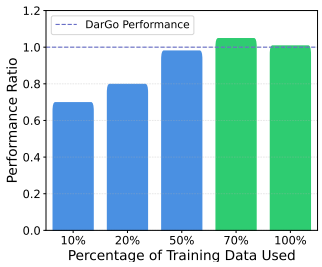


Figure 5: Comparison of DarGO with Lora fine-tuning across varying data size.

3.6 Ablation Study

In Table 6, to better validate the function of each component of DARGO, ablation studies are performed in both Exploration and Inference (More details in Appendix E):

(a) MOI. For the "w/o MOI" scenario, we construct the memory by employing GPT-4o to convert ground-truth code to plans directly, excluding the participation of SLM. This resulted in a significant final performance drop for Phi-3-mini. One explanation is that GPT-4o tends to produce relatively coarse-grained plans (e.g., `"compute percentage of winning"`), whereas SLMs require

more fine-grained steps (e.g., `"compute total number of games"`, `"divide winning games by total wins"`). These higher-level plans can induce hallucinations to SLMs, as they frequently entail additional intermediate steps that increase the risk of errors. Furthermore, without execution-based refinement and calibration through the Plan Optimization (PO) component in MOI, plans generated by GPT-4o often contain inaccuracies that can mislead the SLM, as illustrated in Section 2.2.

(a) Case Study Translation. As elaborated in Section 2.3, case study translation systematically consolidates and refines heterogeneous inputs, including tabular data, user queries, and code samples, into a structured, context-rich representation that SLMs can more effectively process. In the absence of this translation component, the model frequently defaults to errorous or incomplete outputs (e.g., `"SELECT \n\n\n\n..."`), underscoring the difficulty SLMs encounter in performing generalized reasoning when confronted with raw, unprocessed data in DACG.

(c) Functional Plan. A key contribution of our work is the MOI and the use of functional plans to structure and improve the interaction between teacher and student. We evaluate performance by replacing functional plans with textual plans like Chain-of-Thought. The result can prove that our designed functional plans are better orchestration media type compared to general COT plans in data science code generation task.

4 Related Work

Data Analysis Code Generation (DACG). DACG automates code generation for data-centric tasks in formats such as CSV, TSV, and relational databases (RDB) [7, 35]. It requires code that accurately handles schemas, formats, and data semantics, whether in Python for tabular data [9, 10, 47] or SQL for databases [68, 27, 29]. Spreadsheet-based code generation further extends DACG to formula generation in tools [54, 5]. Although large language models have shown promise, privacy remains a challenge in cloud-based environments [35].

Knowledge Distillation. Knowledge distillation can mitigate this problem by transferring LLM capabilities to smaller models, enabling efficient deployment in resource-constrained environments [63]. The field has evolved from early work on softened output training [18] to advanced techniques like task-specific fine-tuning [45], zero-shot learning [57], and instruction-following datasets [58, 57]. Progressive distillation techniques, such as the Orca framework [38], demonstrate the potential for guiding the development of efficient open-source models. Self-distillation approaches have explored autonomous training data generation [58]. Recent advancements have focused on improving the performance and privacy aspects of DACG by knowledge distillation [35]. At the same time, synthetic data has been leveraged to enhance the generalization of SQL generation across different schemas [64]. Even though these techniques are effective, most still require training efforts to transfer knowledge. Our DARGO framework introduces distillation through in-context learning, eliminating the need for task-specific fine-tuning.

Memory for Large Language Models. Memory can enhance LLMs by retaining long-term context and knowledge [69], as in reflection-based frameworks [48], subroutine reuse [50], and self-correction [4]. Complex tasks may require repository-level memory [65, 56, 55]. Typically, memory remains confined to a single model. Our work introduces multi-model memory orchestration (via GPT-4o), enabling smaller models to tap into broader knowledge sources for DACG.

5 Conclusion

In this paper, we presented DARGO, an automatic framework for knowledge distillation from Large Language Models to Small Language Models in Data Science Code Generation. DARGO leverages In-Context Learning to enhance SLM performance without fine-tuning, using model orchestration and memory-based distillation to improve task accuracy. Evaluations on three challenging tabular data analysis datasets that require code generation show a 27.5% relative performance increase for Phi-3-mini. We also show model-agnostic effectiveness, benefiting other SLMS, even they did not participate in the orchestration. These results highlight the potential of DARGO for developing intelligent applications with a focus on privacy and computational efficiency.

6 Acknowledgments

We thank the anonymous reviewers and committees for their helpful comments, suggestions and organizations. We appreciate Xinnuo Xu, Christian Poelitz, Siân Lindley, Hank Lee, Xiaolong Li, Ge Qu for discussion and suggestions. Reynold Cheng, Jinyang Li, were supported by the Research Grant Council of Hong Kong (RGC Project HKU 17202325), the University of Hong Kong (Project 2409100399), and the HKU Faculty Exchange Award 2024 (Faculty of Engineering).

References

[1] Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

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

[3] Rajas Agashe, Srini Iyer, and Luke Zettlemoyer. Juice: A large scale distantly supervised dataset for open domain context-based code generation. *EMNLP-IJCNLP*, pages 5436–5446, 2019. URL https://aclanthology.org/D19-1546.

[4] Arian Askari, Christian Poelitz, and Xinye Tang. Magic: Generating self-correction guideline for in-context text-to-sql. *arXiv preprint arXiv:2406.12692*, 2024.

[5] Kushal Bhatia, Sumit Gulwani, Raksha Saikia, and Samuel Samuel. Sheetcopilot: Assistive table formula completion with neural code generation models. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, pages 701–714, 2023.

[6] Tom B. Brown, Benjamin Mann, Nick Ryder, and Others. Language models are few-shot learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[7] Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

[8] Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W. Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *Trans. Mach. Learn. Res.*, 2023, 2023.

[9] Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *The Twelfth International Conference on Learning Representations*, 2024.

[10] Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Binding language models in symbolic languages. In *The Eleventh International Conference on Learning Representations*, 2023.

[11] Xiang Deng, Yu Gu, Boyuan Zheng, Shijie Chen, Samuel Stevens, Boshi Wang, Huan Sun, and Yu Su. Mind2web: Towards a generalist agent for the web. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=kiYqbO3wqw.

[12] Dujian Ding, Ankur Mallick, Chi Wang, Robert Sim, Subhabrata Mukherjee, Victor Rühle, Laks V. S. Lakshmanan, and Ahmed Hassan Awadallah. Hybrid LLM: Cost-efficient and quality-aware query routing. In *The Twelfth International Conference on Learning Representations*, 2024.

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

[14] Usama M. Fayyad, Gregory Piatetsky-Shapiro, and Padhraic Smyth. From data mining to knowledge discovery in databases. *AI Mag.*, 17(3):37–54, 1996.

[15] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, 2024.

[16] Yu Gu, Yiheng Shu, Hao Yu, Xiao Liu, Yuxiao Dong, Jie Tang, Jayanth Srinivasa, Hugo Latapie, and Yu Su. Middleware for llms: Tools are instrumental for language agents in complex environments. *arXiv preprint arXiv:2402.14672*, 2024.

[17] Jiawei Han, Micheline Kamber, and Jian Pei. *Data Mining: Concepts and Techniques, 3rd edition*. Morgan Kaufmann, 2011. ISBN 978-0123814791.

[18] Geoffrey Hinton. Distilling the knowledge in a neural network. *arXiv preprint arXiv:1503.02531*, 2015.

[19] Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

[20] Xueyu Hu, Ziyu Zhao, Shuang Wei, Ziwei Chai, Qianli Ma, Guoyin Wang, Xuwu Wang, Jing Su, Jingjing Xu, Ming Zhu, Yao Cheng, Jianbo Yuan, Jiwei Li, Kun Kuang, Yang Yang, Hongxia Yang, and Fei Wu. Infiagent-dabench: Evaluating agents on data analysis tasks. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

[21] Harshit Joshi, Abishai Ebenezer, José Cambronero Sanchez, Sumit Gulwani, Aditya Kanade, Vu Le, Ivan Radiček, and Gust Verbruggen. Flame: A small language model for spreadsheet formulas. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 12995–13003, 2024.

[22] Minki Kang, Seanie Lee, Jinheon Baek, Kenji Kawaguchi, and Sung Ju Hwang. Knowledge-augmented reasoning distillation for small language models in knowledge-intensive tasks. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[23] Zixuan Ke, Yijia Shao, Haowei Lin, Tatsuya Konishi, Gyuhak Kim, and Bing Liu. Continual pre-training of language models. In *The Eleventh International Conference on Learning Representations*, 2023.

[24] Mohammad Khanbabaei, Farzad Movahedi Sobhani, Mahmood Alborzi, and Reza Radfar. Developing an integrated framework for using data mining techniques and ontology concepts for process improvement. *J. Syst. Softw.*, 137:78–95, 2018.

[25] Omar Khattab, Arnav Singhvi, Paridhi Maheshwari, Zhiyuan Zhang, Keshav Santhanam, Sri Vardhamanan A, Saiful Haq, Ashutosh Sharma, Thomas T. Joshi, Hanna Moazam, Heather Miller, Matei Zaharia, and Christopher Potts. DSPy: Compiling declarative language model calls into state-of-the-art pipelines. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=sY5N0zY5Od.

[26] Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR, 2023.

[27] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. Kaggledbqa: Realistic evaluation of text-to-sql parsers. In Chengqing Zong, Fei Xia, Wenjie Li, and Roberto Navigli, editors, *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, pages 2261–2273. Association for Computational Linguistics, 2021.

[28] Fangyu Lei, Qian Liu, Yiming Huang, Shizhu He, Jun Zhao, and Kang Liu. S3Eval: A synthetic, scalable, systematic evaluation suite for large language model. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 1259–1286, Mexico City, Mexico, June 2024. Association for Computational Linguistics.

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

[30] Jinyang Li, Nan Huo, Yan Gao, Jiayi Shi, Yingxiu Zhao, Ge Qu, Yurong Wu, Chenhao Ma, Jian-Guang Lou, and Reynold Cheng. Tapilot-crossing: Benchmarking and evolving llms towards interactive data analysis agents. *arXiv preprint arXiv:2403.05307*, 2024.

[31] Aixin Liu, Bei Feng, Bin Wang, Bingxuan Wang, Bo Liu, Chenggang Zhao, Chengqi Dengr, Chong Ruan, Damai Dai, Daya Guo, et al. Deepseek-v2: A strong, economical, and efficient mixture-of-experts language model. *arXiv preprint arXiv:2405.04434*, 2024.

[32] Xiao Liu, Zirui Wu, Xueqing Wu, Pan Lu, Kai-Wei Chang, and Yansong Feng. Are LLMs capable of data-based statistical and causal reasoning? benchmarking advanced quantitative reasoning with data. In *Findings of the Association for Computational Linguistics ACL 2024*, August 2024.

[33] Anton Lozhkov, Raymond Li, Loubna Ben Allal, Federico Cassano, Joel Lamy-Poirier, Nouamane Tazi, Ao Tang, Dmytro Pykhtar, Jiawei Liu, Yuxiang Wei, et al. Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*, 2024.

[34] Pan Lu, Liang Qiu, Kai-Wei Chang, Ying Nian Wu, Song-Chun Zhu, Tanmay Rajpurohit, Peter Clark, and Ashwin Kalyan. Dynamic prompt learning via policy gradient for semi-structured mathematical reasoning. In *International Conference on Learning Representations (ICLR)*, 2023.

[35] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. Wizardcoder: Empowering code large language models with evol-instruct. In *The Twelfth International Conference on Learning Representations*, 2024.

[36] Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. Teaching small language models to reason. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 1773–1781, Toronto, Canada, July 2023. Association for Computational Linguistics.

[37] Yao Mu, Qinglong Zhang, Mengkang Hu, Wenhai Wang, Mingyu Ding, Jun Jin, Bin Wang, Jifeng Dai, Yu Qiao, and Ping Luo. Embodiedgpt: Vision-language pre-training via embodied chain of thought. *Advances in Neural Information Processing Systems*, 36, 2024.

[38] Subhabrata Mukherjee, Arindam Mitra, Ganesh Jawahar, Sahaj Agarwal, Hamid Palangi, and Ahmed Awadallah. Orca: Progressive learning from complex explanation traces of gpt-4. *arXiv preprint arXiv:2306.02707*, 2023.

[39] Yixin Nie, Adina Williams, Emily Dinan, Mohit Bansal, Jason Weston, and Douwe Kiela. Adversarial NLI: A new benchmark for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 4885–4901. Association for Computational Linguistics, 2020.

[40] Taiwo Blessing Ogunseyi, Cossi Blaise Avoussoukpo, and Yiqiang Jiang. A systematic review of privacy techniques in recommendation systems. *International Journal of Information Security*, 22(6):1651–1664, 2023.

[41] Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing of the Asian Federation of Natural Language Processing, ACL 2015, July 26-31, 2015, Beijing, China, Volume 1: Long Papers*, pages 1470–1480. The Association for Computer Linguistics, 2015.

[42] Fabio Petroni, Aleksandra Piktus, Angela Fan, Patrick Lewis, Majid Yazdani, Nicola De Cao, James Thorne, Yacine Jernite, Vladimir Karpukhin, Jean Maillard, Vassilis Plachouras, Tim Rocktäschel, and Sebastian Riedel. KILT: a benchmark for knowledge intensive language tasks. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2523–2544, Online, June 2021. Association for Computational Linguistics.

[43] Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36, 2024.

[44] Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation. *arXiv preprint arXiv:2405.15307*, 2024.

[45] V Sanh. Distilbert, a distilled version of bert: Smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[46] Da Shen, Xinyun Chen, Chenguang Wang, Koushik Sen, and Dawn Song. Benchmarking language models for code syntax understanding. In *Findings of the Association for Computational Linguistics: EMNLP 2022*, pages 3071–3093, Abu Dhabi, United Arab Emirates, December 2022. Association for Computational Linguistics.

[47] Li Shen, Yicong Xu, and Hongbo Wang. Handling ambiguity in natural language code generation. *Transactions of the Association for Computational Linguistics*, 10:113–126, 2022. doi: 10.1162/tacl_a_00414.

[48] Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023.

[49] Ishika Singh, Valts Blukis, Arsalan Mousavian, Ankit Goyal, Danfei Xu, Jonathan Tremblay, Dieter Fox, Jesse Thomason, and Animesh Garg. Progprompt: Generating situated robot task plans using large language models. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11523–11530. IEEE, 2023.

[50] Elias Stengel-Eskin, Archiki Prasad, and Mohit Bansal. Regal: Refactoring programs to discover generalizable abstractions. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*. OpenReview.net, 2024.

[51] Elias Stengel-Eskin, Archiki Prasad, and Mohit Bansal. Regal: Refactoring programs to discover generalizable abstractions. In *Forty-first International Conference on Machine Learning, ICML 2024, Vienna, Austria, July 21-27, 2024*, 2024.

[52] Gemma Team, Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. Gemma 2: Improving open language models at a practical size. *arXiv preprint arXiv:2408.00118*, 2024.

[53] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Online, July 2020. Association for Computational Linguistics.

[54] Daming Wang, Yijun Liu, and Qiang Zhang. Spreadsheet formula generation using lightweight language models. In *Proceedings of the 37th AAAI Conference on Artificial Intelligence*, pages 2109–2116, 2023.

[55] Xingyao Wang, Yangyi Chen, Lifan Yuan, Yizhe Zhang, Yunzhu Li, Hao Peng, and Heng Ji. Executable code actions elicit better llm agents. In *ICML*, 2024.

[56] Xingyao Wang, Boxuan Li, Yufan Song, Frank F Xu, Xiangru Tang, Mingchen Zhuge, Jiayi Pan, Yueqi Song, Bowen Li, Jaskirat Singh, et al. Opendevin: An open platform for ai software developers as generalist agents. *arXiv preprint arXiv:2407.16741*, 2024.

[57] Yizhong Wang, Hamish Ivison, Pradeep Dasigi, Jack Hessel, Tushar Khot, Khyathi Chandu, David Wadden, Kelsey MacMillan, Noah A Smith, Iz Beltagy, et al. How far can camels go? exploring the state of instruction tuning on open resources. *Advances in Neural Information Processing Systems*, 36:74764–74786, 2023.

[58] Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A. Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2023, Toronto, Canada, July 9-14, 2023*, pages 13484–13508. Association for Computational Linguistics, 2023.

[59] Yue Wang, Hung Le, Akhilesh Deepak Gotmare, Nghi DQ Bui, Junnan Li, and Steven CH Hoi. Codet5+: Open code large language models for code understanding and generation. *arXiv preprint arXiv:2305.07922*, 2023.

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

[61] Shitao Xiao, Zheng Liu, Peitian Zhang, Niklas Muennighoff, Defu Lian, and Jian-Yun Nie. C-pack: Packed resources for general chinese embeddings. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, page 641–649, New York, NY, USA, 2024.

[62] Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, et al. Osworld: Benchmarking multimodal agents for open-ended tasks in real computer environments. *arXiv preprint arXiv:2404.07972*, 2024.

[63] Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. A survey on knowledge distillation of large language models. *arXiv preprint arXiv:2402.13116*, 2024.

[64] Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 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)*, Bangkok, Thailand, August 2024.

[65] John Yang, Carlos E Jimenez, Alexander Wettig, Kilian Lieret, Shunyu Yao, Karthik Narasimhan, and Ofir Press. Swe-agent: Agent-computer interfaces enable automated software engineering. *arXiv preprint arXiv:2405.15793*, 2024.

[66] Pengcheng Yin, Wen-Ding Li, Kefan Xiao, Abhishek Rao, Yeming Wen, Kensen Shi, Joshua Howland, Paige Bailey, Michele Catasta, Henryk Michalewski, Oleksandr Polozov, and Charles Sutton. Natural language to code generation in interactive data science notebooks. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Toronto, Canada, July 2023.

[67] Junchi Yu, Ran He, and Zhitao Ying. THOUGHT PROPAGATION: AN ANALOGICAL APPROACH TO COMPLEX REASONING WITH LARGE LANGUAGE MODELS. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=SBoRhRCzM3.

[68] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Tan, Murhaf Fawzi Er, Iris Li, Jun Ma, Zilin Li, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, 2018. doi: 10.18653/v1/D18-1425.

[69] Zeyu Zhang, Xiaohe Bo, Chen Ma, Rui Li, Xu Chen, Quanyu Dai, Jieming Zhu, Zhenhua Dong, and Ji-Rong Wen. A survey on the memory mechanism of large language model based agents. *arXiv preprint arXiv:2404.13501*, 2024.

[70] Zhehao Zhang, Xitao Li, Yan Gao, and Jian-Guang Lou. Crt-qa: A dataset of complex reasoning question answering over tabular data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 2131–2153, 2023.

[71] Boyuan Zheng, Boyu Gou, Jihyung Kil, Huan Sun, and Yu Su. Gpt-4v(ision) is a generalist web agent, if grounded. In *Forty-first International Conference on Machine Learning*, 2024. URL https://openreview.net/forum?id=piecKJ2DlB.

[72] Shuyan Zhou, Frank F Xu, Hao Zhu, Xuhui Zhou, Robert Lo, Abishek Sridhar, Xianyi Cheng, Yonatan Bisk, Daniel Fried, Uri Alon, et al. Webarena: A realistic web environment for building autonomous agents. *arXiv preprint arXiv:2307.13854*, 2023.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: We claim that we are the first work to discuss how well training-free knowledge distillation can achieve.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: We discuss limitation and future work in Appendix G

   Guidelines:

   - The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

   Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: We show instructions of reproducibility in Appendix K

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [Yes]

Justification: We provide the codebase in supplementary material.

Guidelines:

- The answer NA means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [Yes]

   Justification: We provide implementation details in Appendix B.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [Yes]

   Justification: For each inference result, we run experiments 5 times and present average results, which can prove that the performance show statistical significance.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).

- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
- If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We stated that all SLM inference based on one NVIDIA A100 80GB GPU card and how much cost for distillation in Appendix I.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: [Yes]

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: Detailed in Appendix J.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.

- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: We cited them.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.

- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: [NA]

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does not impact the core methodology, scientific rigorousness, or originality of the research, declaration is not required.

Answer: [NA]

Justification: [NA]

Guidelines:

- The answer NA means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
- Please refer to our LLM policy (https://neurips.cc/Conferences/2025/LLM) for what should or should not be described.

A Model Implementation

We implement models for three main categories of purpose:

A.1 Orchestration Models

`gpt-4o`: The Teacher model (`gpt-4o`) is responsible for several key tasks, including Abstraction Lifting (see Section 2.2) and Plan Optimization (see Section 2.2), which are performed while monitoring the performance of the Student model. Additionally, the Teacher model handles the conversion of complex, heterogeneous cases into more readable case studies for Student Learning Models (SLMs), as detailed in Section 2.3.

`Llama-3.3-70B-Instruct`: we use this powerful open-source model as teacher to prove the generalization of DARGO workflow.

`phi-3-mini-128k-instruct`: For the orchestration process, we select this 3.8B parameter SLM as the Student model due to its strong generalization abilities and efficient deployment.

`llama-3.1-8b-instruct`

A.2 Baseline Models

Within the orchestration mode, several families of Student Learning Models (SLMs) are evaluated. These include models from the Phi-3, Starcoder 2, and LlDARGO families:

Phi-3 Family [1]

`phi-3-mini-128k-instruct` (3.8B)

`phi-3-small-128k-instruct` (7B)

`phi-3-medium-128k-instruct` (14B)

Starcoder 2 Family [33]

`starcoder2-7b-instruct`

`starcoder2-15b-instruct`

Llama Family [13]

`codellama-7b-instruct-hf`

`codellama-13b-instruct-hf`

A.3 Models in Knowledge Transmission

In Section 3.4, we explore the knowledge distilled from DARGO to newly developed models, particularly in terms of their ability to generalize knowledge. For this evaluation, we select the following models:

`llama-3.1-8b-instruct`: This model is broad new, yet it shows significant performance improvements when leveraging the distilled knowledge.

`gpt-35-turbo-16k`: We also include a closed-source model in our experiments to demonstrate the effectiveness of our approach across both GPU-deployed and API-based models. Despite its number of parameters is unknown, we consider it as one of SLMs since its performance falls behind of its more advanced versions such as GPT-4.

A.4 Demonstration V.S. Distillation

Given the memory database, we compare the effectiveness of our knowledge distillation techniques, with conventional demonstration-based strategies. In our approach, distillation involves transferring knowledge from the memory database to SLMs through task-specific instructions. On the other hand,

demonstration-based methods guide SLMs by presenting explicit task examples to facilitate analog reasoning [67]. We implemented two variants of few-shot demonstrations: **Static**: Human experts select three representative examples from the memory database, which remain constant across all cases. **Dynamic RAG-based**: Examples are selected from memory database based on similarity to the current query. For fair comparison, we also implement the same RAG system as DarGO-MI, described in Section 3.2.

Our findings show that few-shot demonstration generally underperforms DarGO across datasets. However, RAG-based few-shot demonstration outperforms both our designed knowledge distillation and other baselines on TABMWP. This success appears linked to the simplicity of TABMWP's input data, which averages 2.22 columns and 6.13 rows per data point, with clean numeric or processed string values (Table 7). In contrast, for WIKITQ with irregular value types, and BIRD-SQL with complex schemas, SLMs struggle, generating 38.2% more invalid outputs, such as erroneous SQL queries, in BIRD-SQL. Based on these observations, we conclude that dynamic few-shot demonstration is more convenient and effective for leveraging the memory database when the input data is less complex. On the contrary, for complex data such as tables with dirty values or relational databases, our designed knowledge distillation enables SLMs to better utilize knowledge and perform tasks more effectively.

B Dataset Implementation Details

B.1 Train-Test Distillation Data Statistics

Table 7 summarizes the distributions of the datasets used in this study. These datasets include both single-table and multi-table relational structures, offering a comprehensive aspects for evaluation. The question types range from standard semantic parsing queries to more complex analytical questions involving aggregations and nested operations. This diversity enables a thorough assessment of the model's performance across various query paradigms.

Table 7: Statistics for three datasets. The term `Analysis` indicates that the dataset mainly consists of analytical questions, while `SP` refers to semantic parsing tasks.

STATISTIC	TABMWP	WIKITQ	BIRD-SQL
Dataset Features			
# train examples	1,000	2,000	1,000
# eval examples	1,000	1,000	500
question type	Analysis	SP	SP + Analysis
# toks / Q	26.5	12.6	20.0
Data Structure			
data input type	Single	Single	RDB
# rows / data	6.13	28.5	354k
# columns / data	2.22	6.36	73.3
Code Features			
code type	Python	Python	SQL
answer type	String	String	Code
# toks / code	N/A	N/A	61.15

B.2 Data File Content

For convenient reproduction and following, we preprocess all dataset into more unified data format of `jsonl`. In python task (TABMWP, WIKITQ), each line of data contains `question_id`, `question`, `data_path`, `data_overview`, `answer_type`, `answer`. In SQL task (BIRD-SQL), each line of data contains `question_id`, `question`, `evidence`, `data_path`, `db_id`, `sql`.

B.3 Data Input Content

The main goal of this work is to evaluate the code generation capabilities of models in understanding data schemas and structures across multiple datasets. Given the impracticality of providing all data values in real-world scenarios in which datasets may consist of millions or even billions of rows, we sample values for the part of data input to simulate realistic code generation tasks. We feed the markdown format of schemas with data samples as `data_overview`.

For **TABMWP**, we provide only the column names and the first three rows of values. This enables models to infer the data structure and value types necessary for Python Pandas code generation without exposing all the data.

For **WIKITQ**, which contains more complex and varied value types, we provide the first 10 rows of values and column names to help models navigate the dataset's intricacies.

In the case of BIRD-SQL, which contains relational databases with complex schemas and diverse value types, more advanced schema-linking techniques are often required to retrieve relevant tables

or columns before answering queries [53, 43]. While we consider this advanced schema-linking process as future work for DARGO, our current focus is on the code generation aspect. Therefore, we provide:

- Ground truth retrieved tables, reducing input complexity and simulating realistic human-machine interactions where users might supply potentially relevant tables.
- Full columns with column meaning description files.
- The first three rows of values for each table.

Although the retrieved tables are given, the models must still consider constraints and generate correct SQL queries. As shown in Table 2, performance on Bird-SQL remains relatively low, even with simplified table retrieval, highlighting the challenges of generating accurate SQL queries in complex database environments. This methodology allows us to evaluate code generation capabilities while approximating the real-world challenges of data analysis.

B.4 OOD Cross-Dataset Evaluation Set Statistics.

Table 8 show the data statistics of basic set of unified memory for evaluating DARGO on OOD cross-dataset distillation.

Table 8: Statistics for Basic Set of Unified Memory.

Dataset	# Items
wikitq	1 000
TabMwp	1 000
BIRD-Pandas	300
Juice	1 000

C Action Types

Decomposition. The `Decomposition` action type divides a large, multifaceted step `x` into multiple simpler steps `a` and `b`. By breaking complex workflows into smaller components, the Model Orchestration Interface (MOI) ensures that the resulting plan is both clearly understood and more straightforward for an SLM to execute. This approach clarifies the logic behind each sub-step and allows finer-grained control over how tasks are performed or combined, thereby reducing confusion and facilitating future refinements.

ALT. Sometimes, the instructions for a step can be ambiguous, incorrect, or unnecessarily complex. The `ALT` action type replaces such a problematic step `x` with a newly clarified step `y`. By substituting erroneous or unclear instructions, the MOI ensures that the plan adheres to more accurate logic, minimizing the likelihood of misinterpretation and promoting consistency throughout the workflow.

ADD. When a critical operation is missing or an additional step is required for completeness, the `ADD` action type introduces a new step `a` into the existing plan. Adding steps proves valuable when the plan overlooks essential checks, transformations, or other auxiliary procedures. This mechanism ensures more thorough and dependable solutions.

DELETE. Certain steps can be redundant or risk causing confusion for subsequent code generation. The `DELETE` action type removes any unnecessary step `x`, thereby streamlining the plan. By eliminating irrelevant instructions, the MOI reduces cognitive load on the SLM and maintains a logically consistent sequence of steps that aligns directly with the overarching goal.

SIMPLIFY. Whenever possible, it can be advantageous to simplify complex steps. The `SIMPLIFY` action type replaces a complicated step `x` with a more direct version, `simple_step(x)`. For example, it may transform a solution relying on recursion into an iterative, loop-based approach. Simplification improves both computational efficiency and interpretability, since SLMs often perform better with concise instructions.

Table 9: Performance evaluation of Zero-Shot End-to-End Code Generation, LoRA fine-tuning, and our proposed knowledge distillation techniques on BIRD-SQL. Deeper red shading indicates a larger performance drop compared to the original pre-trained model, while green indicates no decline or improvement.

Model	SIMPLE	MEDIUM	CHALLENGING	OVERALL
Zero-Shot End-to-End Code Gen.				
Original Checkpoint	38.51	21.20	11.76	24.40
LoRA Fine-Tuned	39.86	19.20	10.78	23.60
DARGO *Knowledge Distillation*				
Meta Instruction	**51.35**	**30.40**	16.67	**33.80**

SWITCH. Selecting an appropriate library or tool is critical for achieving both correctness and simplicity. The `SWITCH` action type replaces a step implemented with `packageA` by an equivalent (or more suitable) operation from `packageB`. For instance, one might switch from using `LinearRegression` to `CorrelationCoefficient` when the task is simply to determine the relationship between two variables. This approach avoids unnecessary overhead and preserves clarity in the plan.

By employing these six action types, `Decomposition`, `ALT`, `ADD`, `DELETE`, `SIMPLIFY`, and `SWITCH`, the MOI systematically refines high-level plans. This process results in efficient and easily interpretable workflows, ensuring consistency from the design of the plan to its final implementation by SLMs.

D DARGO Functionality

D.1 MOI Generalization

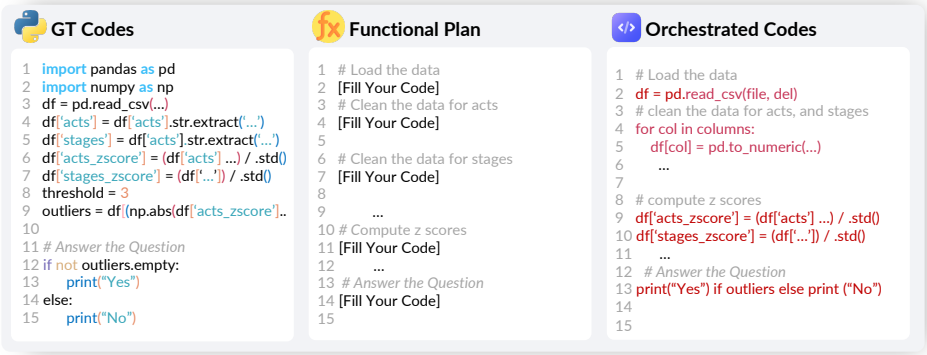


Figure 6: Illustration of how MOI is conducted in Python for Tabular data analysis.

Our Model Orchestration Interface (MOI) is adaptable to different programming languages with different data input settings. Figure 2 shows how MOI is conducted in RDB settings with SQLite, and Figure 6 shows how it's undertaken in Single-tabular data with Python.

D.2 Fine-Tuning v.s. DARGO Knowledge Distillation

E Ablation Study for Inference

We conducted a comprehensive ablation study of DARGO-MI, as shown in Table 10. Code-T5+ is a code embedding model [59], while BGE-Large [61] represents one of the state-of-the-art (SOTA) text embedding models. The study examines two types of RAG Index: one where distance is computed using question embeddings alone, and another where both question and schema embeddings are used. The"Plan + Gen" approach involves first constructing a plan with distilled knowledge, followed by generation using knowledge-driven planning. In contrast, the "Gen" approach involves direct

generation without prior planning. The instruction type labeled `w/ examples` refers to cases where a specific example is provided by the Teacher model. We evaluate performance with 1, 3, and 5 examples to assess the impact of varying numbers of RAG examples. The results of the ablation study reveal several key insights:

Code embeddings outperform text embeddings. The superior performance of Code-T5+ over BGE-Large-en can be attributed to the nature of the task. While text embeddings emphasize on semantic and domain knowledge, code embeddings capture the syntactic and logical structure of coding problems, which is crucial for DACG tasks. Even when presented with identical questions, the code solutions can vary significantly depending on the data input. Code-T5+ is able to effectively embed questions from a programming perspective, benefiting from its pre-trained corpus, whereas text embeddings are less suited for the task.

Embedding only the question is more effective than embedding both the question and schema. The study demonstrates that question-only embeddings lead to better results. This suggests that the inclusion of schema in the embedding may introduce unnecessary complexity, which may hinder performance on the DACG task.

Planning is essential for more complex tasks. The results stress on the importance of planning in a knowledge-driven generation. For tasks requiring complex reasoning, the "Plan + Gen" approach outperforms direct generation (`Gen`), indicating that structured planning significantly improves task performance.

One example may bias the SLM. Involving a single example in the instruction can introduce bias in Small Lanuguage Models (SLMs). A specific example might cause the SLM to over-follow to certain information, leading to hallucinations. For instance, if the example includes a reference to `"singer"`, the SLM may generate plans that include `"singer"` even when the question pertains to an unrelated topic, such as `"cars"`. This observation highlights the lack of robustness in SLMs when exposed to overly specific examples. Consequently, it is better to provide more general, transferable knowledge in instructions. The degraded performance observed with 1 RAG example supports this conclusion, as the model becomes overly reliant on the provided information.

More examples do not always improve performance. Interestingly, increasing the number of RAG examples (from 1 to 5) results in a performance drop. This suggests that longer input sequences may confuse the SLM, making it more difficult to distill relevant knowledge. Based on these findings, we recommend using 3 RAG examples as the optimal balance for complex DACG tasks since it avoids both the biases of a single example and the confusion caused by too many examples.

Table 10: Ablation Study Results of DARGO-MI of Phi-3-mini on BIRD-SQL. The table compares different embedding models, RAG index (with or without schema), reasoning approaches (planning or direct generation), and varying numbers of RAG examples.

Embedding Model	RAG Index	Reasoning Type	Instruction Type	# RAG Examples	Performance
code-t5+	question	plan + gen	no examples	3	33.80
code-t5+	question	gen	no examples	3	31.40 (↓2.40)
bge-large	question	plan + gen	no examples	3	30.00 (↓3.80)
code-t5+	question	plan + gen	w/ examples	3	28.00 (↓5.80)
code-t5+	question+schema	plan + gen	no examples	3	32.40 (↓1.40)
code-t5+	question	plan + gen	no examples	5	31.80 (↓2.00)
code-t5+	question	plan + gen	no examples	1	29.80 (↓4.00)

F Error Analysis

We conducted an error analysis by sampling 50 incorrect cases for both DARGO-MI across three datasets. Although DARGO substantially improves the overall performance of SLMs, we found that 54% of the errors were caused by over-reasoning. This issue tends to emerge even in relatively simple cases. As discussed earlier, SLMs can overly adhere to the instructions derived from planning and guidance, which is problematic when the task is enough simple and does not require decomposition or reasoning. In these cases, direct code generation would lead to more accurate results. The remaining

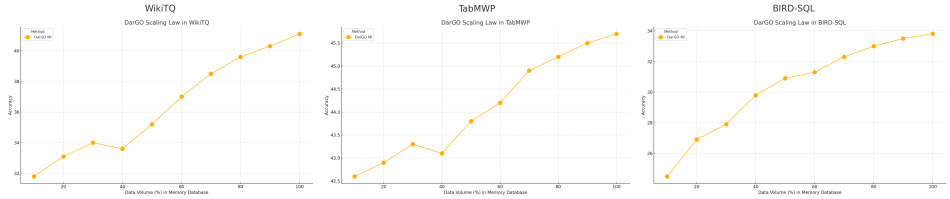


Figure 7: The scaling performance of DARGO on three main DACG datasets.

errors stem from common issues in code generation tasks, such as incorrect string handling, incorrect column selection, database constrain understanding.

G Limitations and Future Work

A key limitation of our current approach with DARGO is the reliance on initial training examples for both LLMs and SLMs to facilitate orchestration. This is why we selected datasets that include a training corpus suitable for distilling knowledge. However, an important avenue for future work is to explore how to generate such training data in a fully zero-shot manner, without relying on human-annotated or enumerated examples. Additionally, as highlighted in the error analysis, over-reasoning negatively impacts performance on simpler tasks, where additional reasoning or decomposition is unnecessary. To address this, future work could focus on developing or prompting smaller models to act as routers, as proposed by Ding et al. [12], to classify questions based on whether they require planning. This would help avoid over-reasoning in straightforward cases and improve the overall efficiency of DARGO.

H Scaling Analysis of DARGO

Figure 7 shows that DARGO scales reliably with increasing data volume. Although its performance is somewhat unstable on very small memory, stability improves rapidly, and accuracy surpasses the baseline once more than 60% of the exploration data are available which proves that the distilled knowledge is being exploited effectively.

I Cost Analysis

We compare the cost of DARGO with other distillation methods shown in Table 12, Table 16 and Table 14. And Table 11, Table 15, Table 13 show the inference cost of DARGO and its comparison with other baselines. We can conclude that: first, it shows DarGO now is the most efficient Distillation work for this task. Second, since inference is conducted by SLMs, the additional token usage is acceptable when considering its significant improvement.

Table 11: BIRD-SQL: Infer (Tab. 16)

Method	**SLM InToks ↓**	**SLM OutToks ↓**	**EX ↑**
End2End (baseline)	358,277	16,751	24.40
Chain-Of-Thought	369,433	19,317	23.80
Static Few-Shot	794,342	17,306	26.20
Dynamic Few-Shot	836,678	26,291	24.40
DSPy	637,146	60,748	22.80
ReGAL	1,602,698	16,045	20.40
DarGO-MI	647,799	64,836	33.80

J Broader Impact

Given DARGO is designed for automating code generation for better and efficient data analysis, it can help data scientists explore the potential risks of financial market, earthquake, which are beneficial to

Table 12: BIRD-SQL: Exploration (Tab. 17)

Method	LLM InToks	LLM OutToks ↓	Cost ↓
DSPy	18,129,504	2,689,076	$108.32
ReGAL	13,259,759	2,381,368	$85.44
DarGO	11,086,194	1,372,421	$62.16

Table 13: WikiTQ: Infer (Tab. 18)

Method	SLM InToks ↓	SLM OutToks ↓	Acc. ↑
End2End (baseline)	235,122	44,033	32.50
Chain-Of-Thought	242,443	149,246	27.70
Static Few-Shot	521,292	45,161	23.00
Dynamic Few-Shot	549,075	93,415	16.60
DSPy	418,131	123,422	26.70
ReGAL	2,082,898	162,949	36.10
DarGO-MI	495,493	168,311	41.10

the society. Also, our study only focuses on code generation, a programming-level language rather than natural language, it will not impact society negatively.

K Reproducibility

We provide codebase in the supplementary files and we list all implementation details in Appendix B And we deliver prompts for each stage and baselines in Appendix L for fully reproducibility.

L Main Prompts

The zero-shot End-to-End Code Generation prompt is shown in Figure 8, Figure 16 and 18 show the zero-shot Chain-Of-Thought reasoning. Figure 19 shows few-shot demonstration prompting. The `few_shot_examples` can be selected by human experts as Static Few-Shot Demonstration, and can be retrieved from DARGO memory database by RAG system as Dynamic Few-Shot Demonstration.

The Figure 8, 9, 10, 11 show prompts for Orchestration between LLMs and SLMs. Figure 12 presents how LLM convert orchestrated successful cases to more understandable case studies to SLMs. LLMs can go through correct cases from memory databases and distill knowledge to an offline and plug-and-plan General Instruction for SLMs to used for new and unseen queries performed by prompts shown in Figure 13 and 14. During inference, SLMs can produce Meta Instructions by prompts in Figure 15. Given distilled knowledge (instructions), SLMs will plan first as shown in Figure 17, and generate codes finally with their knowledge-driven planning, which shows in Figure 18.

M Knowledge Distillation Examples

M.1 Case Study Example

The Figure 20 shows the example of case studies on Python task. Figure 22 present examples of DARGO-MI respectively.

Table 14: WikiTQ: Exploration (Tab. 19)

Method	LLM InToks	LLM OutToks ↓	Cost ↓
DSPy	10,893,180	1,616,141	$65.09
ReGAL	7,967,176	1,431,207	$51.34
DarGO	6,661,181	824,828	$37.35

Table 15: TabMWP: Infer (Tab. 20)

Method	SLM InToks	SLM OutToks	Acc. Score
End2End (baseline)	171,480	38,016	41.80
Chain-Of-Thought	176,764	39,244	41.40
Static Few-Shot	380,177	38,283	36.20
Dynamic Few-Shot	400,358	52,578	52.10
DSPy	304,898	49,073	40.40
ReGAL	767,114	37,679	41.80
DarGO-MI	314,817	125,230	45.70

Table 16: TabMWP: Exploration (Tab. 21)

Method	LLM InToks ↓	LLM OutToks ↓	Cost ↓
DSPy	9,426,638	893,604	$48.75
ReGAL	7,790,084	751,736	$40.49
DarGO	5,330,507	659,862	$29.89

````
You are a data analyst. Given the data, you need to generate the code first to answer the question:

# Please Follow:
- Do not add data inspection in the plan, such as `df.head()` or `print(df.head())` since this is
cheating!
- Do not use any external information.
- The code should be end-to-end, so you cannot encourage yourself to print other things except
  final result. More other information lead to be distracted.

# Question: {question}
# Thought: I need to see the data samples in the first 10 rows:

# Code:
```python
import pandas as pd
df = pd.read_csv('{data_path}', sep='\t')
print(df.head(10))
```

# Observation:
{data_overview}

# Thought: I can generate remaining code to answer this question:
# Code:
```python
import pandas as pd
````

Figure 8: Prompt of baseline end-to-end generation for tasks requiring Python.
````

````
You are a data analyst trainer. You are educating your student to generate right code to answer
tabular data analysis questions. In order to do so, you need to convert your code to code_plan and
let your students to fill to understand plans and analysis. So you cannot generate code by your
own, only generate plans.

# Data Overview at the path {data_path} (first ten rows):
{data_overview}
...

# Question: {question}
# Original Code:
```python
{ground_truth code}
```

You should convert the code into the code_plan format with the placeholder `[FILL YOUR CODE HERE]`:
```code_plan
import ...

# Step 1:....
[Fill Your Code]

# Step 2:....
[Fill Your Code]
...

# Step N: ....
```

Generate your code_plan for your student. DO NOT generate any code by your own. Also ignore and
remove steps of inspecting the data which leads to student cheating.
Please note it's hard for your student to write long code. You will get 1,000 dollars if you have
a good job:
````

Figure 9: Prompt converting ground-truth code to functional plan for python task as example. This is conducted by **LLM Teacher model**.

````
You are a data analyst. Given the data, expert customized functional plan, complete
each line of code to answer questions correctly:

# Please Follow:
- Do not add data inspection in the plan, such as `df.head()` or `print(df.head())`
since this is cheating!
- Do not use any external information.
- The code should be end-to-end, so you cannot encourage yourself to print other
  things except final result. More other information lead to be distracted.

# Question: {question}
# Function Plan:
```python
{functional plan}
```
# Your entire completion code for function plan executable and correct:

# Code:
```python
import pandas as pd
````

Figure 10: Prompt of orchestration coding. This is conducted by **SLM Student model**.
````

```
You are an expert in error analysis and code planning. Your task is to guide your intern in filling out the code for your logic. You need to generate textual plans
as comments that include essential import statements, logics. Currently, the mixed code filled by your intern is incorrect. Then you should analyze and help him.

---------------------------------------------- case begin: ----------------------------------------------
{last turn case}
---------------------------------------------- end ----------------------------------------------

You are experienced data analysis programmer responsible for checking the errors, analyzing the reasons, and helping them correct the code. Note that you cannot
fill the code for them directly. You have four options for actions:
1. **Decomposition(Step Number, new sub steps**: If a step is too complicated and exceeds the intern's capability, decompose this step into multiple smaller steps
for them to fill step by step.
    Actually, you have to decompose steps if there are multiple functions or multiple lines of code in one step since they are not capable!
    step a -> step b, step c
2. **ALT(Step Number, what do you want to alt in detials**: If a step is ambiguous or requires additional information or options, provide an alternative approach
or clarification. But this is a closed-book education, you cannot teach them to use external information aside code and data samples.
    step a -> step b
3. **ADD(Step Number, what do you want to add in details**: If the original step lacks important operations, add a supplementary step to ensure the main code logic
is smooth. But this is a closed-book education, you cannot teach them to use external information aside code and data samples.
   Also all available data are shown, you cannot add or teach them to use `df.head()` to overview data again.
    step a, step c -> step a, step b, step c
4. **DELETE(Step Number, what do you want to delete in details**: If some steps are unnecessary and hinder the intern's understanding of the overall logic, delete
them.
    step a, step b -> step b (deleted step a)
5. **SIMPLIFY(Step Number, simplify specific steps)**: If a step is implemented using recursion and this approach is too complex for the intern to understand or
debug, suggest a non-recursive approach that achieves the same result.
This might involve using iterative methods or other strategies to simplify the logic. If you find code fails due to this, simplify the functions.
step a (recursive) -> step a (iterative)
6. **SWITCH(Step Name, packages to SWITHC)**: If a function relies heavily on a specific package that is known to be complex or not beginner-friendly, suggest
switching to a more intuitive or simpler package that achieves similar functionality. This can help the intern understand the underlying logic without getting
bogged down by the complexities of the original package.
    step a (uses ComplexPackage) -> step a (uses SimplePackage)

You have to provide reasons based on analysis of errors for choosing this action and show your action in <action></action>, then. Finally, you must execute your
chosen action to change original code and fill in the following format:

# format:
Reason:
<reason>...</reason>
Act:
<action>...</action>

# Updated code plan:
```code_plan
import ...

# Step 1:....
[Fill Your Code]

# Step 2:....
[Fill Your Code]
...

# Step N: ....
```

Please note:
- Do not ask students to add data inspection in the code, such as `df.head()` or `print(df.head())`
- Do not guide them to use any external information.
- The mixed code should be end-to-end, so you cannot encourage student to print other things except the final result. More other information would cause student to
be distracted.
- Just focus on how to make students learn how to better plan in the end-to-end code generation.

OK, now change your codes according to your actions.
If you don't follow rules, then you will lose 1 million dollars:
```

Figure 11: Prompt of plan optimization. This is conducted by **LLM Teacher model**.

```
You are a data analyst trainer. You are educating your student to generate pythyon code to answer tabular data analysis questions.

This is a successful case of your code, perform a case study on this:
---------------------------------------------- case begin: ----------------------------------------------
# Question: {question}

# Data Overview at the path {data_path} (first 10 rows):
{data_overview}
...

# Code:
```python
{final orchestrated code}
```
-- case end: --

perform a concise case study! Your case study should only contain

Case Study: [Case Name]
Question: [Question]
Table Info: [Summarized Useful information about Tabular Data]
Objective:
Explanation:

Please note your case study should make your student understand. You don't have to include code again. You will get 1000 dollars if
you have a good job:
```

Figure 12: Prompt of case study conversion. This is conducted by **LLM Teacher model**.

```
You are a data analyst trainer. You are educating your student to generate correct python pandas code to answer tabular
data analysis questions. To test and elicit their knowledge of python pandas code, you generate step-by-step plans that
allow them to fill in code until they succeed.

# These are case studies where they fill the correct code:
---------------------------------------------- case begin: ----------------------------------------------
{case_study_batch}
---------------------------------------------- case end: ----------------------------------------------

# Please note:
- Do not ask students to add data inspection in the code, such as `df.head()` or `print(df.head())`
- Do not guide them to use any external information.
- The mixed code should be end-to-end, so you cannot encourage student to print other things except the final result. More
other information would cause student to be distracted.
- Just focus on how to make students learn how to better plan in the end-to-end code generation.

According to the previous case studies, analyze and reflect how to generate plans which can make your student fill the
correct code. Summarize 5-7 key points.
```

Figure 13: Prompt of aggregation prompt of each batch of case studies. This is conducted by **LLM Teacher model**.

```
You are a data analyst trainer. You are educating your student to generate correct python pandas code to answer tabular
data analysis questions. To test and elicit their knowledge of code, you generate step-by-step plans that allow them to
fill in subqueries until they succeed.

# These are case studies where they fill the correct code:
------------------------------------------- case begin: ------------------------------------------------
{last layer of case studies}
------------------------------------------- case end: --------------------------------------------------

# Please note:
- Do not ask students to add data inspection in the code, such as `df.head()` or `print(df.head())`
- Do not guide them to use any external information.
- The mixed code should be end-to-end, so you cannot encourage student to print other things except the final result. More
other information would cause student to be distracted.
- Just focus on how to make students learn how to better plan in the end-to-end code generation.

Following case studies, please summarize 5-7 key points about how to plan and generate correct code to answer the tabular
data analysis questions accurately.
Students will take your notes directly.

# You need to start with:
```1. You should
```

Figure 14: Prompt of summarization prompt of batch of case studies in the last layer. This is conducted by **LLM Teacher model**.

```
You are a data analysis trainer. Your are teaching your student to plan and generate python code
accurately. You find some case study for reference.

# There are case studies:
------------------------------------------- case begin: ------------------------------------------------
{case_studies}
------------------------------------------- case end: --------------------------------------------------

Following case studies, please summarize key 5-7 points about how to plan and generate correct python
code to answer the data analysis questions accurately.

# You will use them to educate your student:
```successful plan suggestions:
1. You Should
```

Figure 15: Prompt of in-time summarization for meta-instructions. This is conducted by **SLM Student model**.

```
You are a data engineer. Given the sample data, generate python code plan to answer the question
accurately.

# Please Follow:
- Do not add data inspection in the plan, such as `df.head()` or `print(df.head())` since this is
cheating!
- Do not use any external information.
- The code should be end-to-end, so you cannot encourage yourself to print other things except final
  result. More other information lead to be distracted.
```

Question: {question}
Thought: I need to see the data samples in the first 10 rows:

Code:
```python
import pandas as pd
df = pd.read_csv('{data_path}', sep='\t')
print(df.head(10))
```

Observation:
{data_overview}

Thought: I should have a step-by-step text plan for generating this code first. I will fill my plan
into the template in details:
```code_plan
Step 1: ...
Step 2: ...
...
Final Step: ...
```

Generate your plan step by step for the question:

Let's think step by step:
```code_plan
Step 1:
```

Figure 16: Prompt of generating Chain-Of-Thought. This is conducted by **SLM Student model**.

```
You are a data engineer. Given the sample data, generate python code plan to answer the question

# Please Follow:
- Do not add data inspection in the plan, such as `df.head()` or `print(df.head())` since this is
cheating!
- Do not use any external information.
- The code should be end-to-end, so you cannot encourage yourself to print other things except final
  result. More other information lead to be distracted.


# There are some important successful plan suggestions from experts:

```successful plan suggestions:
{successful_plan_suggestions}
```

Question: {question}
Thought: I need to see the data samples in the first 10 rows:

Code:
```python
import pandas as pd
df = pd.read_csv('{data_path}', sep='\t')
print(df.head(10))
```

Observation:
{data_overview}

Thought: Referring to [successful plan suggestions], I should have a step-by-step text plan for
generating this code first. I will fill my plan into the template in details:
```code_plan
Step 1: ...
Step 2: ...
...
Final Step: ...
```

Generate your plan step by step for the question:

Let's think step by step:
```code_plan
Step 1:
```
```

Figure 17: Prompt of knowledge-driven planning. This is conducted by **SLM Student model**.

```
You are a data engineer. Given the sample data, generate python code to answer the question accurately.

# Question: {question}
# Thought: I need to see the data samples in the first 10 rows:

# Code:
```python
import pandas as pd
df = pd.read_csv('{data_path}', sep='\t')
print(df.head(10))
```

Observation:
{data_overview}

Thought: I can generate code to answer this question and print the result. I will fill my code in the
template:
```python
[Your Code]
```

Let's think step by step for the question:
{step-wise plans}

Code:
```python
import pandas as pd
```
```

Figure 18: Prompt of code generation given step-wise planning. This is conducted by **SLM Student model**.

```
You are a data analyst. Given data sample, you need to generate pandas code first to answer the question.

Generate your pandas code to answer the question, and print the result for your to understand. Fill your
code in
```python
[Your Code]
```

Please follow:
- Do not add data inspection in the plan, such as `df.head()` or `print(df.head())` since this is
cheating!
- Do not use any external information.
- The code should be end-to-end, so you cannot encourage yourself to print other things except the final
result. More other information would cause sutdent to be distracted.

There are some examples:
--------------------------- Examples Start ---------------------------
{few_shot_examples}
--------------------------- Examples END ---------------------------

Question: {question}
Thought: I need to see the data samples in the first 10 rows:

Code:
```python
import pandas as pd
df = pd.read_csv('{data_path}', sep='\t')
print(df.head(10))
```

Observation:
{data_overview}

Thought: I can generate code to answer this question:

Code:
```python
import pandas as pd
```
```

Figure 19: Prompt of few-shot demonstration. This is conducted by **SLM Student model**.

Case Study: Average Weight Calculation for Specific Players

Question:
What is the average weight of Jamarr Sanders and Robert Williams?

Table Info:
- **Columns**: Name, Height, Weight (lbs.), Position, Class, Hometown, Previous Team(s)
- **Sample Data**:
  - Jamarr Sanders: Weight 210 lbs.
  - Robert Williams: Weight 210 lbs.

Objective:
To calculate the average weight of the players Jamarr Sanders and Robert Williams from the given dataset.

Explanation:
1. **Load Data**: The data is loaded from a tab-separated values (TSV) file.
2. **Filter Data**: Rows corresponding to the names "Jamarr Sanders" and "Robert Williams" are filtered from the dataset.
3. **Calculate Average**: The average weight of the filtered rows is computed.
4. **Output**: The result is printed as an integer.

By following these steps, the student can understand how to filter specific rows in a dataset and perform calculations on the filtered data. This case demonstrates the practical application of data manipulation and analysis using pandas in Python.

Figure 20: Example of case studies for tasks requiring Python. This is conducted by **LLM Teacher model**.

1. You should **break down the task into manageable steps**. Each step should build on the previous one, guiding you through the process logically.

2. You should **emphasize data handling and cleaning**. This includes handling missing values, normalizing case, and ensuring data consistency."

3. You should **focus on filtering and extraction**. Guide yourself on how to filter and extract relevant data based on specific criteria. This is often the core of the analysis."

4. You should **perform aggregation and counting**. Learn how to perform aggregation operations like counting, summing, or finding minimum/maximum values to derive insights from the data."

5. You should **present the result clearly**. Ensure that the final step involves presenting the result in a clear and concise manner. This reinforces the importance of communicating findings effectively."

6. You should **avoid distractions**. Keep the instructions focused on the end-to-end process without encouraging unnecessary intermediate outputs or external information. This helps maintain your focus on the task at hand."""

Figure 21: Example of General Instruction for tasks requiring Python. This is conducted by **LLM Teacher model**.

"question": "which country rank last?"

1. Understand the problem statement and the data structure.

2. Load the data using appropriate libraries (e.g., pandas).

3. Perform necessary data manipulation and cleaning.

4. Identify the relevant columns and values for the analysis.

5. Use appropriate functions and methods to filter, sort, and extract the required information.

6. Output the result in a clear and concise manner.

Figure 22: Example of General Instruction for tasks requiring Python. This is conducted by **SLM Student model** in time.