Demonstrating TOFFEE: A Learned System for Synthesizing Data Agent Trajectories at Scale

Ziting Wang
Nanyang Technological University
Singapore
ziting001@e.ntu.edu.sg

Yin Li
Nanyang Technological University
Singapore
yin010@e.ntu.edu.sg

Zuhao Yang
Nanyang Technological University
Singapore
YANG0756@e.ntu.edu.sg

Xiuchang Li
Huawei
China
lixiuchang@huawei.com

Jiale Bai
Industrial and Commercial Bank of China Limited, China
baijl2@sdc.icbc.com

Gao Cong
Nanyang Technological University
Singapore
gaocong@ntu.edu.sg

ABSTRACT

LLM-powered data agents are playing an increasingly important role in data-driven decision making. However, existing data agents struggle to generalize to unseen data environments and analytical workflows, especially in heterogeneous enterprise settings. This creates a growing need for synthesizing high-quality data agent trajectories that capture complex analytical workflows for given data environments. Such trajectories support two key downstream uses: they can serve as supervised finetuning (SFT) data that adapts data agent models to the target domain, and as in-context learning (ICL) demonstrations to guide general-purpose LLMs in unfamiliar data environments. Thus, we introduce TOFFEE, a system for synthesizing high-quality data agent trajectories from given data environments via Monte Carlo Tree Search (MCTS) with adaptive model selection and cross-task prefix reuse. We show that TOFFEE can effectively generate scalable trajectory data for complex analytical tasks across heterogeneous environments. In this demonstration, we present the system framework of TOFFEE, including its task pool construction, trajectory explorer, and learned cost model. We also introduce the web interface of TOFFEE and its workflow, and demonstrate two end-to-end scenarios: trajectory synthesis for data agent finetuning, and demonstration-augmented data agent reasoning.

PVLDB Reference Format:
Ziting Wang, Yin Li, Zuhao Yang, Xiuchang Li, Jiale Bai, and Gao Cong. Demonstrating TOFFEE: A Learned System for Synthesizing Data Agent Trajectories at Scale. PVLDB, 19(12): 4738 - 4741, 2026.
doi:10.14778/3827998.3828110

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/wang0702/toffee.

Proceedings of the VLDB Endowment, Vol. 19, No. 12 ISSN 2150-8097.
doi:10.14778/3827998.3828110

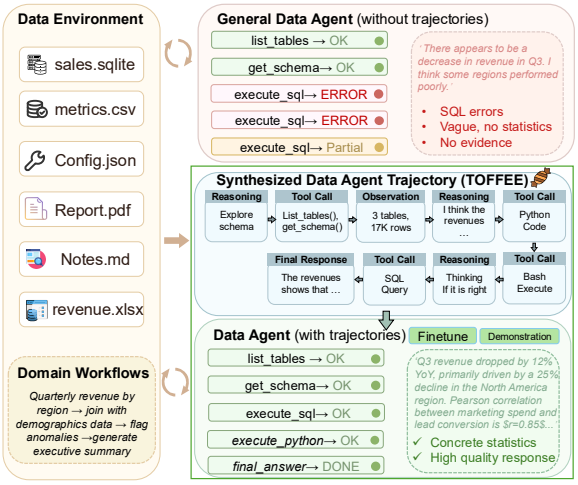


Figure 1: Data agent trajectory synthesis. Top: without trajectories, the agent produces SQL errors and vague conclusions. Middle: synthesized trajectories. Bottom: trajectory-augmented agent via SFT or ICL.

1 INTRODUCTION

Data agents that analyze data by interleaving reasoning with tool execution (*e.g.,* SQL, Python) have attracted growing attention for data-driven decision-making [1, 2]. Correspondingly, major data lake and data warehouse platforms are beginning to integrate such capabilities, *e.g.,* Databricks Genie, Snowflake Cortex Analyst, and BigQuery data agents. However, existing data agents struggle to generalize to unseen data environments and analytical workflows [1], especially in heterogeneous enterprise environments. Synthesizing high-quality data agent trajectories, i.e., multi-step sequences of reasoning, tool invocations, and execution results, for a given data environment can bridge this gap. Figure 1 illustrates this. The top row shows a data agent operating without trajectories: it makes repeated SQL errors and returns a vague conclusion. The middle row shows synthesized trajectories generated by running agent actions against the actual data environment. When used for SFT or ICL (as shown in the bottom row), these trajectories help the same agent avoid errors and produce concrete evidence (*e.g.,*

Q3 revenue dropped 12% YoY, Pearson r=0.85). However, acquiring such trajectories at scale remains difficult [3, 4]. Single-pass generation discards all progress upon any step failure, while best-of-N sampling incurs redundant computation across independent attempts. Building such a system faces the following challenges.

Challenges. First, data agent trajectory synthesis requires a large pool of diverse analytical tasks, *e.g.,* "compute quarterly revenue by region" or "correlate marketing spend with lead conversion rates," each grounded in the tables and files of the target data environment. Prompting LLMs to generate tasks from schemas often produces trivial or unsolvable questions, and manually curating tasks does not scale. The first challenge is *how to automatically construct realistic, solvable tasks from user-specified data environments* (**C1**). Second, synthesizing data agent trajectories is inherently complex: each step requires a full cycle of LLM inference and tool execution and can choose among tool actions (*e.g.,* SQL query, Python script, or schema discovery) generated by different LLMs, so the number of possible trajectories grows exponentially with the number of steps. The second challenge is *how to synthesize high-quality trajectories in such a vast search space* (**C2**). Third, each synthesis step requires choosing an inference configuration: which LLM to invoke, how many prior steps to feed as prompt context, and whether to enable extended thinking of LLMs. For example, a smaller model can list tables and read column names, but fails when the step requires writing a multi-table join query; a larger model handles the join correctly but wastes budget on such routine steps. The third challenge is *how to select the right inference configuration at each step based on the step's difficulty* (**C3**).

Our Method. To address these challenges, we introduce TOFFEE, a budget-aware trajectory synthesis system. First, to automatically construct a large pool of analytical tasks, we design a *Task Pool Construction* pipeline. It discovers dependencies in the data environment through lightweight scripts, then reverses each dependency into a multi-step analytical task, and verifies that each task is both solvable and meaningful (addressing C1). Second, for a task, to efficiently navigate the vast search space of trajectory synthesis, we design a *Trajectory Explorer*. It implements MCTS-based search, where every candidate step is executed against the data environment, naturally capturing diverse execution paths (*e.g.,* error recoveries) (addressing C2). Third, to select the right inference configuration (*e.g.,* which LLM to call) at each step based on the step's difficulty, we design a *Learned Cost Model* (LCM). The LCM observes execution rewards and learns state-dependent policies through online reward learning, allowing it to distinguish routine steps from complex analytical steps without manual tuning (addressing C3).

Preliminary results support this design. Under the same per-task budget, TOFFEE synthesizes higher-quality trajectories than single-pass generation and best-of-N sampling. Models finetuned on the synthesized trajectories achieve substantial gains over their base models on KramaBench and DSBench (Section 3).

In this demonstration, TOFFEE's interactive web interface lets users connect heterogeneous data environments, synthesize trajectories with real-time search tree visualization, and export results for SFT and ICL. We present two end-to-end scenarios.

2 OVERVIEW OF TOFFEE

The architecture of TOFFEE is presented in Figure 2. TOFFEE takes as input a data environment $\mathcal{E}$, which may contain databases, files, and resources accessible through sandboxed code. It produces data agent trajectories in three stages: task pool construction (left), trajectory exploration (center), and export for SFT or ICL (right).

Task Pool Construction. Data agent trajectory search is useful only when tasks are realistic, solvable, and tied to the data environment. A naive way is to directly prompt the schema into an LLM, but due to this method's limited context of the database, it often yields trivial or unsolvable questions. To address this, TOFFEE constructs a data agent task pool before search, grounding each task in the data environment itself. As shown on the left of Figure 2, TOFFEE builds tasks from the given data environment. The environment is either connected by the user or sampled from existing benchmarks such as Spider, LiveSQLBench, and SpreadsheetBench. It first profiles the environment to collect schema metadata and sample rows, then runs predefined SQL templates to discover data dependencies such as a shared customer ID linking several tables, or order cancellation rates that vary sharply across regions, and finally prompts an LLM to phrase each discovered dependency as a meaningful question for a data agent to answer. Each dependency is executed before its question is written, and the final result is recorded as the task's answer key. TOFFEE then verifies each candidate task: it replays the solution steps the way an agent would, reruns each query to confirm the results are stable, and rejects questions that an LLM can answer without accessing the data. It admits only tasks with a deterministic answer and verified execution evidence, ensuring strict isolation from downstream benchmarks to prevent evaluation contamination.

Trajectory Explorer. The *Trajectory Explorer* synthesizes trajectories for each task through a tree search: each node in the search tree is an analysis state, and each edge is a tool invocation, such as running a SQL query or a Python script, generated by a specific LLM configuration. Rather than committing to a single path, the Explorer expands multiple branches and keeps the most promising. The search starts from a shared prefix reused across tasks in the same data environment, so common operations such as schema discovery and data profiling run only once. At each expansion, the Explorer invokes the LCM, described next, to choose the branching width and inference configuration under the remaining budget. When a step hits an execution error, the Explorer does not discard the partial trajectory; it branches into a repair path, and the resulting error-diagnosis-repair sequence is retained as training data that teaches downstream agents to recover from realistic mistakes. A trajectory is stored once a deterministic verifier confirms that its executed outputs reproduce the evidence recorded at task construction and its final answer matches the execution-verified answer key.

Learned Cost Model. The *Learned Cost Model* (LCM) is a budget-aware controller for the Explorer that selects the LLM and inference configuration for each expansion from the current analysis state and the remaining per-task API budget. For instance, it routes routine schema discovery to a small, fast model while escalating complex multi-table reasoning to a more capable one. As shown

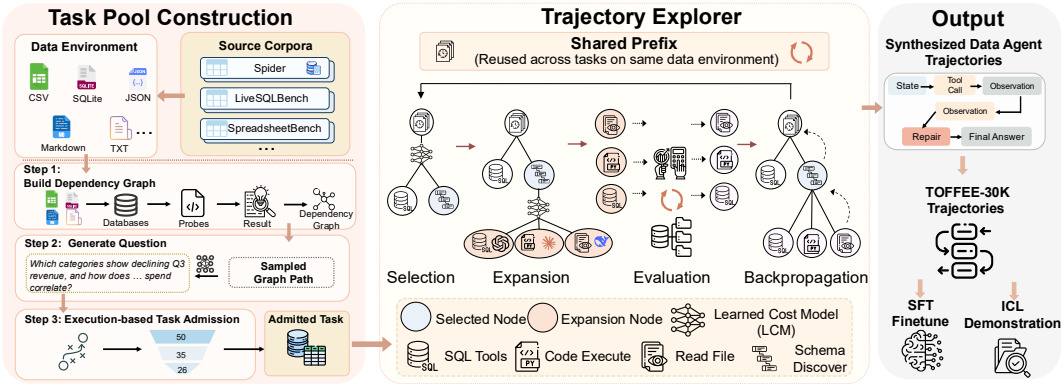


Figure 2: System overview of TOFFEE: task pool construction (left), trajectory explorer (center), and export of data agent trajectories for SFT or Demonstration ICL (right).

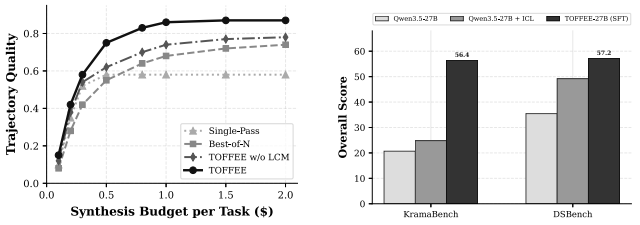
at the bottom of Figure 2, at each expansion the LCM extracts a state feature vector encoding trajectory progress, schema coverage, and recent error history, then scores each feasible configuration, including the operator, model, context length, and reasoning effort, by combining the predicted reward μ with an exploration bonus σ for less-explored configurations. It sets the branching width K from the disagreement among top-ranked configurations, narrowing K as the budget shrinks. The Explorer and LCM form a closed feedback loop: after each step, the system computes a reward from execution success and analytical progress, and updates the LCM's bandit parameters via ridge regression, improving its routing and branching over time.

Trajectory Export and Downstream Usage. As shown on the right of Figure 2, each trajectory is a sequence of (state, tool call, observation) tuples, exported in two modes. For *SFT*, trajectories become multi-turn training samples that teach agent models the full analytical workflow, yielding models such as TOFFEE-9B and TOFFEE-27B. For *ICL demonstration*, the highest-quality trajectories serve as worked examples: when a user poses a question, the system retrieves a similar trajectory into the LLM prompt, so the model follows the demonstrated workflow on the unfamiliar environment.

3 PRELIMINARY EXPERIMENTAL RESULTS

We conduct experiments on an Ubuntu server, where synthesis workers use API calls to LLMs via OpenRouter. Trajectory Quality is scored on a 0 to 1 scale from execution signals (*e.g.,* step success rate, analytical progress) and a deterministic check of the final answer against each task's execution-verified answer key. Best-of-N uses N=5; all methods in Figure 3b share the same agent prompt and tool access. Downstream evaluation uses KramaBench [2] and DSBench [1], with TOFFEE-9B and TOFFEE-27B finetuned from Qwen3.5-9B and Qwen3.5-27B.

Figure 3a compares trajectory quality under varying per-task budgets. Single-Pass plateaus once the budget covers a single full attempt. Best-of-N improves with more budget but with diminishing returns, as each independent attempt repeats work from scratch. TOFFEE w/o LCM improves over them through MCTS exploration but is limited by uniform configuration at every step.



(a) Trajectory quality vs. budget. **(b) Downstream score.**

Figure 3: Preliminary results on synthesis efficiency and downstream effectiveness.

TOFFEE consistently outperforms all baselines, converging to high quality at lower cost, which confirms the value of adaptive per-step model selection via the LCM. Figure 3b shows the downstream effectiveness of TOFFEE trajectories applied to Qwen3.5-27B. Using TOFFEE-synthesized trajectories as ICL demonstrations already yields improvement over the zero-shot baseline. Finetuning on TOFFEE trajectories (TOFFEE-27B) further amplifies the gain, more than doubling the zero-shot score on KramaBench.

4 DEMONSTRATION SCENARIO

The demonstration includes a *Synthesis* mode for trajectory production (Figure 4) and an *Inference* mode for demonstration-augmented data agent reasoning (Figure 5).

4.1 End-to-End Experience

Figure 4 shows the Synthesis mode of TOFFEE. Users perform trajectory synthesis in the following four steps.

Step 1: Environment Setup. Users connect a data environment with heterogeneous sources such as databases, CSV files, JSON, and Markdown documents. The *Data Environment* panel on the top-left of Figure 4 then lists its table names, row counts, and schema.

Step 2: Synthesis Configuration. In the *Synthesis Configuration* panel on the top-right of Figure 4, users assemble a model pool spanning three cost tiers, namely *Cost-Effective*, *Capable*, and *Premium*. They

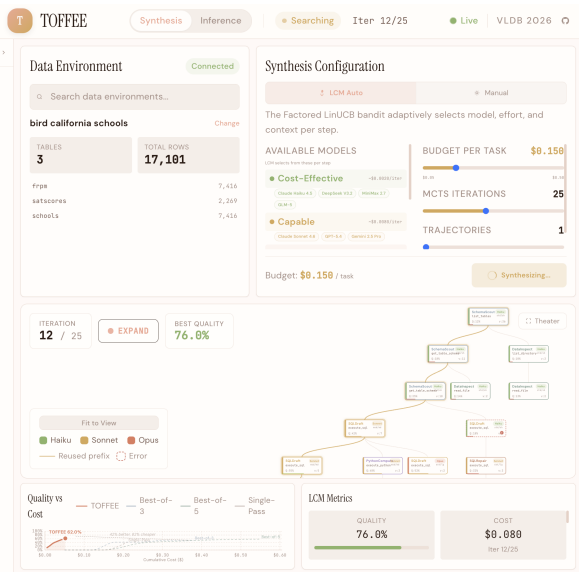


Figure 4: Synthesis mode of TOFFEE.

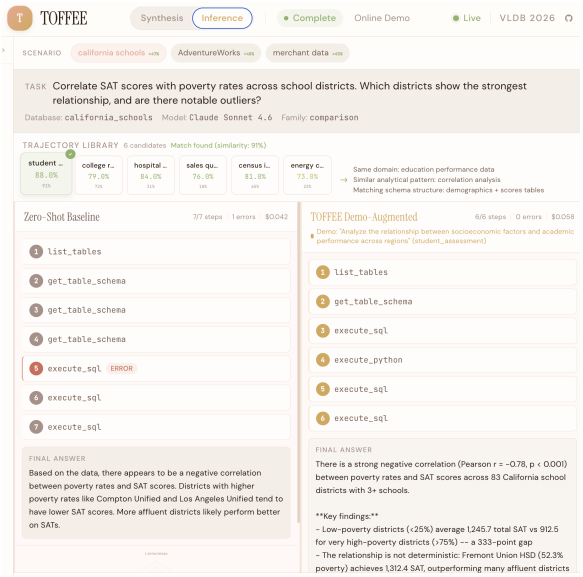


Figure 5: Inference mode of TOFFEE.

choose *LCM Auto* mode, where the LCM selects the model, reasoning effort, and context length per step, or *Manual* mode for direct control, and set the per-task budget, MCTS iteration count, and target trajectory count via sliders.

Step 3: Trajectory Synthesis. The MCTS tree visualization in the middle of Figure 4 renders each node as a card showing the tool type, the color-coded model tier selected by the LCM, and the evaluation score. For instance, a schema reading node stays in the Cost-Effective tier, while an SQL repair node after an execution error escalates to the Capable or Premium tier, reflecting the LCM's policy of investing more on harder steps. The *Quality vs Cost* chart tracks quality over iterations against baselines such as Best-of-N and Single-Pass, and the *Metrics* panel shows cost and progress.

Step 4: Export and Downstream Use. After synthesis, users export trajectories as SFT samples or test them as ICL demonstrations in the *Inference* tab (Figure 5).

4.2 Scenario 1: Trajectory Synthesis for Finetuning

In Figure 4, the user connects a school database with 3 tables and 17K rows, configures a three-tier model pool, and sets the per-task budget and MCTS iteration count. The system profiles the environment, constructs the task pool, and launches MCTS-based search. As synthesis proceeds, the tree visualization grows live, with execution-error nodes spawning repair branches and the shared prefix reused across tasks on the same database. Clicking a node shows the query executed at that step and the result it returned. The user then exports the synthesized trajectories as SFT training samples for data agent models such as TOFFEE-9B and TOFFEE-27B. The exported samples follow the standard multi-turn reasoning format and can be loaded directly by common finetuning frameworks.

4.3 Scenario 2: Demonstration Augmented Data Agent Reasoning

As shown in Figure 5, in the *Inference* tab, the user selects a scenario and poses an analytical task such as "Correlate SAT scores with poverty rates across school districts and identify outliers."

The system maintains a library of 30K trajectories synthesized across hundreds of data environments. Given the new task, it retrieves the most relevant trajectories ranked by domain and workflow similarity, and injects them as few-shot demonstrations. Figure 5 shows a 91% similarity match from an education performance database, guiding the agent through a proven workflow.

The interface presents a side-by-side comparison: the zero-shot baseline produces a vague answer with execution errors, while the TOFFEE-augmented agent follows a structured workflow, including schema discovery, table joins, and statistical computation, yielding concrete findings (*e.g.,* Pearson r=−0.78, p<0.001) with zero errors.

ACKNOWLEDGMENTS

This research is supported by Singapore MOE AcRF Tier-2 grant MOE-T2EP20223-0004.

REFERENCES

[1] Liqiang Jing, Zhehui Huang, Xiaoyang Wang, Wenlin Yao, Wenhao Yu, Kaixin Ma, Hongming Zhang, Xinya Du, and Dong Yu. 2025. DSBench: How Far Are Data Science Agents from Becoming Data Science Experts?. In *ICLR*.

[2] Eugenie Lai, Gerardo Vitagliano, Ziyu Zhang, Sivaprasad Sudhir, Om Chabra, Anna Zeng, Anton A. Zabreyko, Chenning Li, Ferdi Kossmann, Jialin Ding, Jun Chen, Markos Markakis, Matthew Russo, Weiyang Wang, Ziniu Wu, Michael J. Cafarella, Lei Cao, Samuel Madden, and Tim Kraska. 2026. KramaBench: A Benchmark for AI Systems on Data-to-Insight Pipelines over Data Lakes. In *ICLR*.

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

[4] Xiaotian Lin, Yanlin Qi, Yizhang Zhu, Themis Palpanas, Chengliang Chai, Nan Tang, and Yuyu Luo. 2025. LEAD: Iterative Data Selection for Efficient LLM Instruction Tuning. *Proc. VLDB Endow.* 19, 3 (2025), 426–439.