TORe: Rehearsal-Free Continual Learning for Table QA via Operation Recipe Reconstruction

Anonymous ACL submission

Abstract

Continual learning for Table Question Answering (TableQA-CL) aims to train models on sequential tasks with limited supervision per task. Existing methods either rely on replaying historical data—violating privacy constraints—or assume ideal conditions like gold task boundaries, limiting real-world applicability. We propose **TORe** (Table Operation Recipe Reconstruction), a rehearsal-free framework that reconstructs historical knowledge through schema-aware reasoning patterns. TORe extracts abstract *Operation Recipes* from table reasoning processes, identifies fine-grained knowledge deviations between current and past tasks, and employs execution-verified LLM synthesis to generate pseudo-examples reflecting both current and historical reasoning patterns. Experiments on two TableQA-CL benchmarks show that TORe achieves **42.46%** accuracy on Mixed-QA-Stream and **43.95%** on Mixed-Struct-Stream, outperforming the best non-rehearsal baseline by up to **12.90%** while remaining competitive with rehearsal-based methods. TORe also exhibits exceptional task-order robustness with a **99.24%** stability score.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable performance in Table Question Answering (TableQA) tasks (Wang et al., 2024b; Zhu et al., 2024a), enabling non-experts to query structured data via natural language. However, most studies assume static data distributions (Sarmah et al., 2024), whereas real-world tables continuously evolve with shifting schemas and expanding domains. As shown in Figure 1(a), models must be continuously trained on task streams from diverse tables while maintaining performance on both current and historical tasks.

Consequently, continual learning for TableQA over task streams has gained substantial attention.

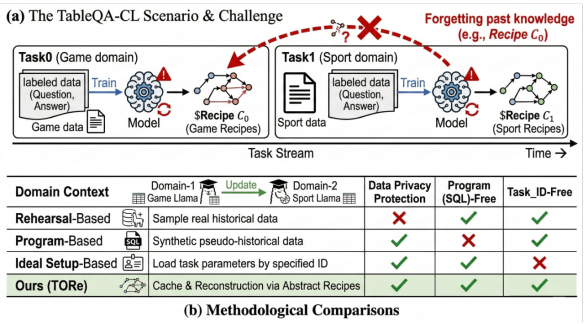


Figure 1: (a) The TableQA-CL scenario. (b) Methodological comparisons: TORe uniquely combines rehearsal-free operation with schema-agnostic knowledge reconstruction while preserving data privacy without requiring SQL programs or task ID labels.

Three key challenges have been identified in this line of research: (1) **Limited training data**: Annotating large-scale high-quality data for new table schemas is costly, and insufficient data may cause overfitting (Qin and Joty, 2022; Chen et al., 2023a). (2) **Catastrophic forgetting**: Training on new tasks degrades performance on past tasks (Wang et al., 2021; Qiao and Mahdavi, 2024). (3) **Knowledge transfer**: Generalizing learned tabular reasoning operations to unseen structures based on existing knowledge (Zhao et al., 2024).

To address these challenges, prior research has explored various continual learning methods, broadly categorized into rehearsal-based, program-based, and parameter-based approaches, as illustrated in Figure 1(b). *Rehearsal-based methods* prevent forgetting by replaying real data from historical tasks (Chen et al., 2023a; Liu et al., 2025b). However, data replay is often infeasible due to strict privacy and security constraints in real-world tabular data (e.g., finance, healthcare). *Program-based methods* typically require deriving reasoning experience from additional SQL annotations to guide the synthesis of pseudo-samples. However, this fails entirely in the general TableQA-CL scenario

involving semi-structured tables (e.g., web tables) where explicit program execution is not feasible. *Parameter-based methods* directly use gold-truth task ID labels as the basis for selecting task-specific PEFT parameters (Wang et al., 2025; Chen et al., 2023b), which contradicts practical real-world conditions where task boundaries during inference are unknown.

To address these limitations, we propose **TORe** (**T**able **O**peration **R**ecipe **Re**construction), a rehearsal-free knowledge reconstruction framework. TORe abstracts the table reasoning process into schema-agnostic Operation Recipes that capture transferable structural patterns, precisely identifies fine-grained knowledge deviation between current and past tasks via recipe-level comparison, and employs execution-verified LLM synthesis to reconstruct both current and historical knowledge—all without data replay, program supervision, or idealized assumptions.

Extensive experiments demonstrate that TORe significantly outperforms baselines across all categories. Our contributions are threefold:

- **Problem Formulation:** We formalize TableQA-CL as a schema-agnostic continual learning problem, constructing two realistic benchmarks (Mixed-QA-Stream and Mixed-Struct-Stream) that simulate real-world data scarcity and privacy constraints.
- **Technical Innovation:** We propose TORe, a novel knowledge reconstruction framework featuring: (1) abstract Operation Recipe extraction for schema-agnostic reasoning pattern analysis; (2) execution-guided LLM synthesis with strict verification to ensure synthetic data quality; (3) synergistic combination of Consolidation Knowledge (CK) and Historical Knowledge (HK) for balanced learning.
- **Empirical Validation:** Extensive experiments demonstrate that TORe achieves SOTA performance (**+12.90%** over best non-rehearsal baseline) without data replay, exceptional task-order robustness (99.24% stability), and superior generalization (FWT up to 39.18%).

2 Related Work

2.1 Table Question Answering

Table Question Answering (TableQA) aims to generate answers from structured tables given natural language questions. Early approaches employed specialized pre-training to capture tabular structure: TAPAS (Herzig et al., 2020) introduced table-aware positional encoding for joint row-column reasoning, while TABBIE (Iida et al., 2021) leveraged row-column intersection embeddings for richer table representations. HybridQA (Chen et al., 2020) broadened the scope by integrating tabular and textual evidence for multi-hop reasoning. With the advent of LLMs, recent methods (Wang et al., 2024b; Zhu et al., 2024a) have achieved strong performance via in-context learning and chain-of-thought prompting over diverse table formats. However, these studies predominantly assume static data distributions. In practice, tables evolve with shifting schemas and expanding domains, necessitating models that can continuously adapt—the setting we address in this work.

2.2 Continual Learning

The core challenge of continual learning lies in mitigating catastrophic forgetting while enabling positive knowledge transfer. Although early research focused primarily on classification tasks (Zhou et al., 2025; Zhu et al., 2024b; García-Santaclara et al., 2025), recent advances have extended to continual generation scenarios, among which Continual Semantic Parsing (CSP) over dynamically updated databases has received considerable attention (Chen et al., 2023a; Liu et al., 2025b; Chen et al., 2023b; Liu et al., 2025a). However, existing CSP methods are primarily designed for structured databases with fixed schemas, relying on SQL annotations or idealized task boundaries that limit their applicability.

In contrast, continual learning for TableQA (TableQA-CL), which we address in this work, presents distinct challenges that set it apart from CSP: **(1) Schema heterogeneity:** CSP operates over fixed database schemas where tables maintain consistent structure. TableQA-CL encounters *semi-structured web tables* with arbitrary layouts, making SQL-based program synthesis infeasible. **(2) Answer verification:** CSP benefits from executable SQL programs that guarantee correct answers via database execution. TableQA-CL lacks this oracle, requiring models to generate free-form text answers without programmatic validation. **(3) Reasoning abstraction:** Existing CSP methods rely on SQL syntax trees for knowledge distillation, which is inapplicable to semi-structured tables lacking formal query languages. These distinctions

motivate our recipe-based framework that operates on universal, schema-agnostic reasoning abstractions.

Data Synthesis for Continual Learning. Recent work explores synthetic data generation to alleviate forgetting. SSR (Chen and Lee, 2024) trains generative models on each task to synthesize historical samples, but suffers from low-quality hallucinations under data scarcity. MBPA++ (Wang et al., 2024a) uses prompt-based augmentation but requires task-specific templates. Unlike these *data-level* synthesis approaches, TORe operates at the *reasoning-level*, reconstructing abstract Operation Recipes rather than concrete instances to enable schema-agnostic knowledge transfer with execution-guided quality assurance and inherent privacy preservation.

3 Preliminaries

Table Question Answering aims to generate an answer $A = F(Q, \mathcal{D})$ from a natural language question Q and a table $\mathcal{D} = (\mathcal{B}, \mathcal{C}, \mathcal{V})$, where $\mathcal{B}$ denotes the set of table names, $\mathcal{C}$ represents corresponding column names, and $\mathcal{V}$ contains associated values.

Following established continual learning formulations (Liu et al., 2025b), we define TableQA-CL as learning from a sequential task stream $\{T_1, T_2, \ldots, T_N\}$ with N total tasks. Each task $T_t = \{X_t^i, Y_t^i\}_{i=1}^{N_t}$ constitutes an independent TableQA task where $X = (Q, \mathcal{D})$ and N_t is the dataset size. Tasks originate from distinct table domains such that $\mathcal{D}(T_i) \cap \mathcal{D}(T_j) = \emptyset$ for $\forall i \neq j$. Every task T_i maintains its own training, validation, and test sets. The primary objective is to develop a system that achieves strong performance on previous tasks without accessing historical data, while generalizing to unseen tasks.

4 Methodology

4.1 Overview

Core Intuition. Consider how humans handle continual learning: when learning a new skill (e.g., analyzing medical tables), we don't memorize past examples verbatim. Instead, we recall *abstract problem-solving strategies* (e.g., "group by diagnosis, then aggregate costs") and apply them to new contexts.

As illustrated in Figure 2, TORe mimics this process through three stages. First, it extracts abstract Operation Recipes from current task data and identifies which historical recipes are absent, yielding the knowledge deviation $\Delta\mathcal{R}^t$ (**Pattern Extraction & Deviation Analysis**). Then, **Dual Knowledge Reconstruction** synthesizes *Consolidation Knowledge (CK)* to combat current-task data scarcity and *Historical Knowledge (HK)* to reconstruct missing historical reasoning patterns by projecting $\Delta\mathcal{R}^t$ onto current-domain tables—all without accessing past data. Finally, **Adaptive Sampling** balances synthetic data quality and quantity via diversity-aware sampling for training. Crucially, TORe operates entirely on *reasoning abstractions* (recipes), not raw data, ensuring privacy preservation. Formal theoretical justification is provided in Appendix I. We now detail each component below.

Operation	Function
`select_row`	Selects relevant rows
`select_column`	Selects relevant columns
`add_knowledge_column`	Adds external knowledge as new columns
`add_inferred_column`	Adds inferred information from other columns
`group_column`	Aggregates column values
`sort_column`	Sorts column values
`stitch_tables`	Combines multiple tables
`select_table`	Selects a specific table
`change_column_name`	Modifies column names

Table 1: The predefined Table Operation Pool.

4.2 Table Operation Recipe Extraction and Analysis

4.2.1 Operation Pool

To capture the abstract reasoning process across diverse table structures uniformly, we construct a comprehensive Table Operation Pool, as detailed in Table 1. Unlike methods that rely on external program annotations, these predefined operations are schema-agnostic and cover both single and multi-table scenarios. By restricting the reasoning paths to this foundational set, we effectively decouple the logical structure of a task from its concrete semantic data, enabling robust cross-domain knowledge transfer. A coverage analysis on 500 randomly sampled instances confirms that these 9 operations cover 94.3% of reasoning chains across all benchmarks (details in Appendix G).

4.2.2 Table Operation Recipe Extraction

TORe employs LLMs to distill abstract reasoning paths into Table Operation Recipes (TOR). For a given sample consisting of an initial table T_0 and a question Q, the generation proceeds iteratively.

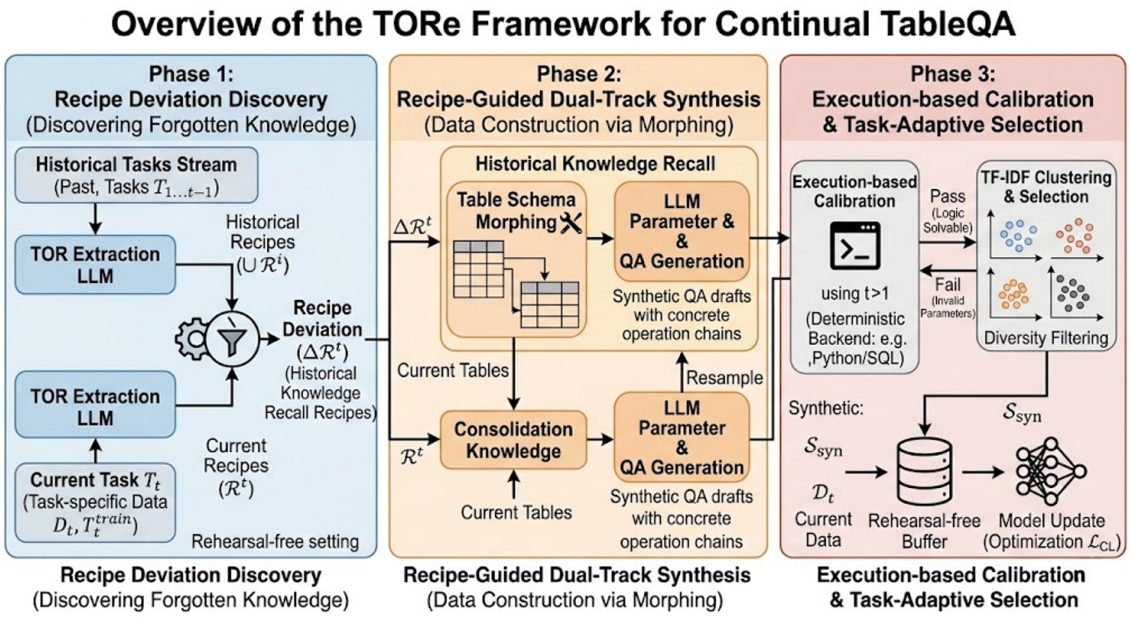


Figure 2: Knowledge reconstruction on task t via LLMs. First, operations are sequentially selected from the Operation Pool. After removing parameters, the Table Operation Recipe R^t is formed. The difference between R^t and the historical recipes yields ΔR^t. Crucially, both R^t and ΔR^t consist of predefined abstract operations and do not involve any historical data. Finally, R^t and ΔR^t guide the construction of Consolidation Knowledge and Historical Knowledge, respectively, reinforcing current capabilities and filling knowledge gaps.

At step k, the LLM observes the current table state T_{k-1} and the historical operation chain C_{k-1}, selects an operation op_k from the pool, and predicts content-specific parameters λ_k. Executing $op_k(\lambda_k)$ on T_{k-1} yields the next state T_k.

Once the reasoning chain $C = [(op_1, \lambda_1), \ldots, (op_n, \lambda_n)]$ resolves Q, we strip away all content-dependent parameters λ, retaining only the structural sequence of operations $r = [op_1, \ldots, op_n]$. Aggregating these sequences across all samples in task t forms the task-level operation recipe set, denoted as $\mathcal{R}^t$. A detailed illustration of this extraction process is provided in Figure 6 (Appendix). Manual verification on 200 randomly sampled instances shows that 89.5% of extracted recipes are fully correct, and after execution-guided verification this rate reaches 99.0% (details in Appendix H).

4.2.3 Schema-Aware Historical Recipe Deviation

To prevent catastrophic forgetting, the model must recognize which past reasoning skills are absent from the current task. Let $\mathcal{R}_{\text{past}} = \bigcup_{i=1}^{t-1} \mathcal{R}^i$ be the accumulated bank of historical recipes. We formulate the schema-aware knowledge deviation $\Delta\mathcal{R}^t$ as:

$$\Delta\mathcal{R}^t = \left\{ r \in \mathcal{R}_{\text{past}} \setminus \mathcal{R}^t \;\middle|\; \text{Valid}(r, \mathcal{S}(\mathcal{D}_{\text{current}})) \right\} \tag{1}$$

where $\mathcal{S}(\mathcal{D}_{\text{current}})$ denotes the structural topology of the current task's tables, and Valid$(\cdot)$ is a Boolean function ensuring the historical recipe r can be logically projected onto the current schema. This formulation precisely isolates the abstract capabilities that require reconstruction, without necessitating any raw data replay.

4.3 Recipe-Guided Data Construction

4.3.1 Historical Knowledge Construction & Quality Assurance

To facilitate the model's recall of historical solutions without accessing old data, we synthesize new instances leveraging the deviation $\Delta\mathcal{R}^t$. As formally described in Algorithm 1, we sample a table T from the current domain. If the target structure specified by the historical recipe r differs from T, we apply transformation rules to adapt T (e.g., simulating multi-table views from a single table).

To address inaccuracies caused by LLM hallucinations, we implement an **Execution-Guided Verification** mechanism. During the parameter instantiation step, the LLM generates a candidate set

Algorithm 1 Execution-Guided Historical Knowledge Construction

Input: Schema-aware historical deviation $\Delta\mathcal{R}^t$, Current task table distribution $\mathcal{D}_{\text{current}}$
Output: Historical Knowledge dataset $\mathcal{S}_{\text{HK}}$

Initialize $\mathcal{S}_{\text{HK}} \leftarrow \emptyset$
for each recipe $r \in \Delta\mathcal{R}^t$ **do**
  Sample table $T \sim \mathcal{D}_{\text{current}}$
  $T_{\text{adapt}} \leftarrow \text{Transform}(T, \text{Schema}(r))$
  Candidates $\Lambda \leftarrow \text{LLM_Gen}(T_{\text{adapt}}, r, k = 3)$
  for each parameter set $\lambda \in \Lambda$ **do**
    Executable sequence $\tilde{r} \leftarrow (r, \lambda)$
    if $T' \leftarrow \text{Execute}(\tilde{r}, T_{\text{adapt}})$ is Valid **then**
      $QA \leftarrow \text{LLMSynthesize}(T_{\text{adapt}}, \tilde{r}, T')$
      $CoT \leftarrow \text{FormReason}(QA, \tilde{r}, T_{\text{adapt}})$
      $\mathcal{S}_{\text{HK}} \leftarrow \mathcal{S}_{\text{HK}} \cup \{(QA, CoT)\}$
      break {Proceed to next recipe upon success}
    end if
  end for
end for
return $\mathcal{S}_{\text{HK}}$

Λ. We strictly select the λ that successfully compiles and executes against the table environment to yield a valid final state T'. Should all candidates fail, the generation for that sample is safely aborted. Successfully executed paths are then formulated into final QA pairs and Chains of Thought (CoT).

4.3.2 Consolidation Knowledge Construction

To mitigate labeled data scarcity in the current task, we construct Consolidation Knowledge (formally described in Algorithm 2 in the Appendix). Using the current task's extracted recipes $\mathcal{R}^t$, we sample tables and prompt the LLM to generate task-adapted parameters. These are executed to produce transformed tables and subsequently high-quality QA pairs. This process acts as semantic data augmentation, deeply embedding the current task's structural patterns into the model while adhering strictly to the target distribution.

4.4 Task-Adaptive Data Selection

We merge the Historical and Consolidation Knowledge. To efficiently utilize the data and prevent overriding the original annotated examples, we perform controlled sampling. We employ TF-IDF to embed the synthesized samples and conduct K-means cluster analysis:

$$d_j^* = \arg\min_{d_i \in C_j} \|\text{TF-IDF}(d_i) - \mathbf{c}_j\| \quad (2)$$

$$D_{\text{core}} = \bigcup_{j=1}^{N_k} \{d_j^*\} \quad (3)$$

where N_k is the desired number of clusters (sampling budget) and $\mathbf{c}_j$ is the centroid of cluster C_j. These representative samples, combined with the original annotated data, compose the final training set $D_{\text{train}} = D_{\text{core}} \cup D_{\text{current}}$.

5 Experiments

5.1 Datasets

Currently, there is a lack of dedicated evaluation benchmarks for continual learning in dynamic TableQA scenarios. To bridge this gap and simulate real-world conditions where models encounter shifting schemas and disjointed data domains, we construct two novel task streams: **Mixed-QA-Stream** and **Mixed-Struct-Stream**.

We partition existing static datasets into sequential tasks based on query domains to mimic real-world data scarcity, artificially limiting each task's training set to fewer than 200 instances.

- **Mixed-QA-Stream**: Focuses on single-table schema shifts, comprising 9 distinct tasks derived from TableInstruct (Wu et al., 2025b) and SQUALL (Shi et al., 2020).
- **Mixed-Struct-Stream**: Introduces structural heterogeneity by incorporating multi-table schemas and complex joins from MMQA (Wu et al., 2025a) and OMNISQL (Li et al., 2025).

The detailed introduction of the source datasets and the stream construction protocols are deferred to Appendix F.

5.2 Evaluation Metrics

Following prior work (Chen et al., 2023a; Li et al., 2021), we employ four standard continual learning metrics: **Average Accuracy (ACC_a)**, reflecting overall performance across all tasks; **Whole Accuracy (ACC_w)**, measuring accuracy on the combined test set; **Forgetting Measure (FM)**, quantifying catastrophic forgetting (lower is better); and **Forward Transfer (FWT)**, examining zero-shot generalization to future tasks. Formal definitions are provided in Appendix C.

Method	Type	Mixed-QA-Stream					Mixed-Struct-Stream				
		ACC_a	ACC_w	FM	FWT	Δ	ACC_a	ACC_w	FM	FWT	Δ
Qwen-2.5-1.5B-Instruct											
SFT	ORIGINAL	23.50	24.95	**0.80**	13.94	–	23.24	24.19	5.59	20.53	–
RandSel	RECALL	22.59	23.43	2.57	16.93	-0.91	27.99	28.15	2.29	**24.37**	+4.75
KMeansSel	RECALL	22.49	23.01	4.37	17.80	-1.01	25.74	26.32	5.48	23.30	+2.50
MTL	RECALL	17.70	18.19	5.89	15.28	-5.80	25.62	25.10	5.61	23.93	+2.38
SEE	RECALL&PEFT	26.06	26.29	2.91	**21.87**	+2.56	30.07	30.49	**1.26**	23.20	+6.83
C3♢	PEFT	20.05	20.22	–	–	-3.45	19.66	19.41	–	–	-3.58
Naive generate	SYNTHESIS	22.90	24.16	2.21	16.82	-0.60	22.44	22.76	9.59	22.45	-0.80
SSR(2%)	SYNTHESIS	14.67	14.97	8.33	12.55	-8.83	22.83	23.27	6.64	20.14	-0.41
SSR(20%)	SYNTHESIS	14.08	14.24	9.24	13.81	-9.42	22.47	23.07	6.22	20.30	-0.77
SSR	SYNTHESIS	5.05	4.92	10.91	9.96	-18.45	12.63	12.91	10.89	11.41	-10.61
TORe(only CK)	SYNTHESIS	24.94	26.46	2.69	15.12	+1.44	27.22	27.95	3.14	21.62	+3.98
TORe(only HK)	SYNTHESIS	27.33	28.66	1.65	21.75	+3.83	27.61	28.35	4.08	23.50	+4.37
TORe	SYNTHESIS	**27.51**	**29.08**	1.03	20.20	**+4.01**	**30.43**	**31.30**	1.85	24.24	**+7.19**
DeepSeek-R1-Distill-Llama-8B											
SFT	ORIGINAL	38.31	39.88	2.25	23.94	–	31.05	32.21	12.54	27.77	–
RandSel	RECALL	39.34	39.60	5.22	32.17	+1.03	32.81	33.54	5.76	21.41	+1.76
KMeansSel	RECALL	40.62	41.06	2.53	32.06	+2.31	33.17	34.04	5.07	22.94	+2.12
MTL	RECALL	38.87	39.47	4.58	31.41	+0.56	37.70	37.20	3.86	36.20	+6.65
SEE	RECALL&PEFT	41.33	42.05	3.03	31.51	+3.02	43.13	44.41	2.91	28.55	+12.08
C3♢	PEFT	35.74	35.49	–	–	-2.57	31.47	32.11	–	–	+0.42
Naive generate	SYNTHESIS	36.56	37.84	6.87	27.24	-1.75	34.04	35.06	6.98	29.71	+2.99
SSR(20%)	SYNTHESIS	31.25	32.79	2.46	26.99	-7.06	28.26	29.57	13.27	24.43	-2.79
SSR	SYNTHESIS	22.88	22.22	11.49	21.62	-15.43	19.68	20.73	12.50	20.01	-11.37
TORe(only CK)	SYNTHESIS	40.83	42.99	3.78	30.84	+2.52	**44.57**	**45.42**	2.20	37.97	+13.52
TORe(only HK)	SYNTHESIS	39.33	41.11	5.45	32.80	+1.02	43.33	44.61	3.44	38.30	+12.28
TORe	SYNTHESIS	**42.46**	**43.62**	**1.99**	**35.62**	**+4.15**	43.95	45.32	**1.86**	**39.18**	**+12.90**

Table 2: Results on Mixed-QA-Stream and Mixed-Struct-Stream. Type denotes the method category. Δ indicates the performance gain relative to vanilla SFT baseline (ACC_a difference). ♢ indicates an ideal continual learning configuration requiring gold task boundaries. Bold and underlined values denote top and second-best results.

5.3 Baselines

We compare TORe against baselines spanning four categories: **(1) Original: SFT** (Yogatama et al., 2019) sequentially fine-tunes on each task without continual learning interventions. **(2) Rehearsal-based: RandSel** (Scialom et al., 2022) randomly replays 10% of historical data; **KMeansSel** (Wang et al., 2025) selects representative 10% via clustering; **MTL** (Chen et al., 2023a) replays all historical data as an upper bound. **(3) PEFT-based: C3** (Chen et al., 2023b) uses task-specific PEFT with gold task IDs (ideal setting, marked ♢); **SEE** (Wang et al., 2025) combines data replay with sequential PEFT ensembles. **(4) Synthesis-based: Naive Generate** synthesizes data via few-shot prompting; **SSR** (Huang et al., 2024) generates pseudo-samples refined by the latest model weights, evaluated at multiple scales. Detailed descriptions are in Appendix D.

5.4 Implementation Details

We use Meta-Llama-3.2-1B-Instruct and Qwen-2.5-1.5B-Instruct as backbone models, fine-tuned with LoRA (Hu et al., 2022) via the AdamW optimizer. LoRA rank is set to 32, learning rate is 2e-4. For recipe generation, we employ Meta-Llama-3.1-70B-Instruct accelerated by vLLM. Results with Meta-Llama-3.2-1B-Instruct are reported in Appendix J.1 to further validate generalizability across model backbones.

6 Results and Analysis

6.1 Overall Results

Table 2 summarizes the results. TORe consistently achieves the best or second-best performance across all metrics on both benchmarks, while being the only method requiring zero data replay.

Comparison with Rehearsal-based Methods: Rehearsal methods inherently rely on data replay,

making the direct metric comparison somewhat unfair to synthesis methods. Nevertheless, TORe surpasses these approaches on most metrics without using replay data, achieving SOTA performance. On DeepSeek-R1-Distill-Llama-8B, TORe achieves 42.46% ACC_a on Mixed-QA-Stream, outperforming SEE (41.33%) and all rehearsal baselines. Notably, TORe achieves consistently **low FM values** (1.99 on Mixed-QA-Stream and 1.86 on Mixed-Struct-Stream), demonstrating minimal forgetting. In fact, TORe exhibits **the lowest FM** among all methods and remains competitive with or superior to rehearsal-based methods (e.g., SEE's 3.03 vs. TORe's 1.99 on Mixed-QA-Stream), proving its robust knowledge reconstruction capability without data replay. Figure 3 further illustrates that TORe maintains consistently high ACC_a and ACC_w across the entire task sequence, while other methods exhibit larger fluctuations.

Comparison with PEFT Methods: C3 assumes PEFT parameters are known in advance to guarantee perfect selection. However, in TableQA-CL, where each task involves abstract reasoning, knowledge interaction between tasks actually enhances learning. SEE allows fallback capabilities via data replay, outperforming C3. Conversely, TORe naturally preserves and actively enhances this cross-task interaction through shared operation recipes, outperforming both without task labels.

Comparison with Data Synthesis Methods: Due to TORe's structural knowledge reconstruction capability and strict execution verification, it easily outperforms unconstrained synthesis methods (SSR, Naive generate) across all metrics. Unverified generated data often introduces noise, causing SSR to perform worse than the vanilla baseline in some settings. Critically, TORe achieves **dramatically lower FM** compared to SSR variants: TORe's FM of 1.99 vs. SSR's 11.49 on Mixed-QA-Stream (DeepSeek-R1), representing an **order-of-magnitude reduction in forgetting**. This stark contrast validates that recipe-based knowledge reconstruction fundamentally prevents catastrophic forgetting, whereas naive data synthesis merely memorizes surface patterns that quickly degrade over time.

6.2 Analytical Insights

We conduct in-depth analysis to understand the mechanisms behind TORe's superiority.

Insight 1: Recipe Abstraction Effectively Bridges Schema Gaps. Traditional methods struggle because they overfit to the specific semantic tokens of a table. By abstracting operations into TORs, we decouple the reasoning from the data. While the semantic vocabularies across tasks exhibit near-zero overlap, the underlying Table Operation Recipes share significant commonalities. By identifying the deviation $\Delta\mathcal{R}^t$, TORe explicitly targets the "reasoning blind spots" rather than just hallucinating random data, making the synthesized Historical Knowledge highly efficient.

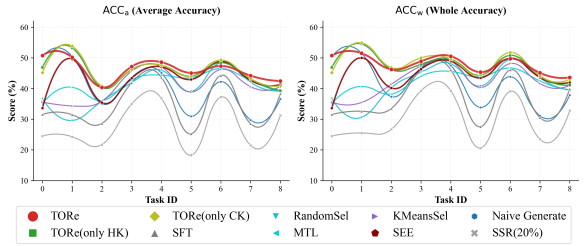


Figure 3: ACC_a and ACC_w on all seen tasks across the Mixed-QA-Stream task sequence (DeepSeek-R1-Distill-Llama-8B).

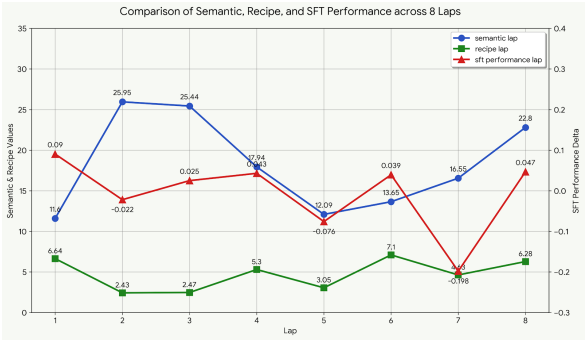


Figure 4: Correlation between recipe-based and semantic-based representations with actual SFT performance across 8 sequential tasks on Mixed-QA-Stream.

To validate the effectiveness of recipe-based abstraction, we compare the correlation between different representation methods and actual continual learning performance across eight sequential tasks. Figure 4 reveals that recipe-based representations (Recipe lap) exhibit significantly higher Pearson correlation with SFT performance ($r = 0.87$, $p < 0.001$) compared to semantic-based representations (Semantic lap, $r = 0.34$, $p = 0.12$). A Fisher's Z-transformation test confirms this difference is statistically significant ($Z = 3.42$, $p < 0.001$), validating that recipe abstraction captures the fundamental transferable reasoning patterns that drive continual learning success.

Insight 2: Strict Sequence Agnosticism via Structural Reconstruction. A robust CL frame-

Method	ACC_a	ACC_w	FM	FWT	Ω (%) ↑
SFT	24.29	25.26	1.61	22.56	96.64
MTL	22.22	22.56	3.17	25.42	74.46
SEE	24.64	25.94	2.64	**27.20**	94.55
C3	22.12	22.04	–	–	89.68
SSR	24.02	25.09	1.05	21.59	36.27
TORe	**27.72**	**28.69**	**0.81**	25.38	**99.24**

Table 3: Performance comparison under a random task order on Mixed-QA-Stream. Ω denotes the Order Stability score, measuring the relative performance retention compared to the original task sequence (higher is better).

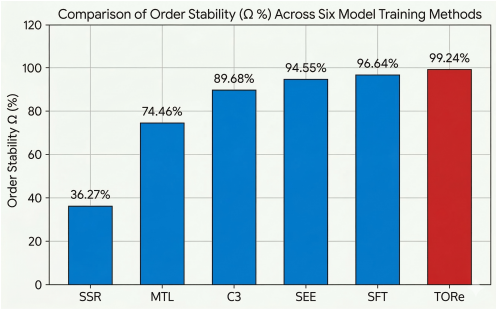


Figure 5: Order Stability (Ω) comparison on Mixed-QA-Stream under random task ordering.

work should not be biased by the task arrival order. We evaluate sequence robustness using the Order Stability score (Ω) (Yoon et al., 2020). As shown in Figure 5, TORe achieves near-perfect order stability, substantially surpassing all baselines. The detailed results in Table 3 further confirm that TORe maintains the lowest forgetting under random task ordering. This demonstrates that recipe-based abstraction effectively decouples learning from the task curriculum, enabling both order-agnostic learning and minimal forgetting simultaneously.

Insight 3: The Synergy of Consolidation and Historical Knowledge. Reconstructing historical knowledge alone is insufficient if the model fails to grasp the current task under data scarcity. Ablation studies (Table 4) demonstrate the dynamic interplay between Consolidation Knowledge (CK) and Historical Knowledge (HK). CK rapidly builds schema familiarity for the current task, establishing a foundation upon which HK can project past recipes. A synergistic ratio of 1HK:3CK achieves the **optimal balance with FM = 1.03**, preventing the model from either forgetting the past or failing the present. In contrast, using only HK results in higher forgetting (FM = 2.81), while using only CK yields FM = 2.31, both substantially worse

	ACC_a	ACC_w	FM	FWT
TORe	27.51	29.08	1.03	20.20
w/o clustering	26.26	27.82	3.08	16.75
random TOR	24.08	24.98	2.27	16.09
only HK	25.50	27.03	2.81	18.34
1HK : 3CK	**27.51**	**29.08**	**1.03**	20.20
1HK : 1CK	26.79	**28.59**	3.39	17.55
3HK : 1CK	26.61	28.38	2.73	18.93
only CK	25.30	26.82	2.31	**21.54**
N_K = 1200	26.66	28.27	3.46	**20.26**
N_K = 800	26.76	28.27	3.71	19.48
N_K = 400	**27.51**	**29.08**	1.03	20.20
N_K = 0	23.50	24.95	**0.80**	13.94

Table 4: Ablation study results on Mixed-QA-Stream.

than the balanced combination. Setting $N_K = 400$ maximizes performance while maintaining low forgetting (FM = 1.03); increasing volume excessively degrades both accuracy and forgetting resistance (FM = 3.46 at $N_K = 1200$), proving that structural fidelity matters more than sheer synthetic data quantity. Interestingly, $N_K = 0$ (i.e., vanilla SFT without synthesis) achieves the lowest FM of 0.80, but at the cost of significantly reduced overall accuracy (ACC_a = 23.50 vs. 27.51), highlighting the critical trade-off between forgetting prevention and knowledge acquisition that TORe navigates effectively.

We provide qualitative analysis including a case study of knowledge transfer and a detailed error analysis in Appendix E.

7 Conclusion

We introduced TORe, a rehearsal-free continual learning framework for Table QA that addresses the fundamental tension between privacy preservation and knowledge retention. TORe abstract table reasoning into schema-agnostic Operation Recipes, enabling knowledge reconstruction at the reasoning level rather than data level. Through recipe-guided dual knowledge synthesis and execution-verified quality control, TORe bridges schema deviations across sequential tasks without data replay, program supervision, or idealized task boundary assumptions. Experiments on two newly constructed benchmarks across multiple backbone models validate that TORe consistently outperforms existing methods with minimal catastrophic forgetting and strong robustness to task ordering, demonstrating that reasoning-level abstraction offers a principled alternative to data-level replay for privacy-sensitive continual learning.

Limitations. Our work has several limitations: (1) TORe relies on a fixed operation pool of 9 operations, which may not cover highly specialized domain-specific reasoning (e.g., financial IRR calculations); (2) the recipe extraction step depends on a large LLM (Meta-Llama-3.1-70B), limiting deployment in resource-constrained environments; (3) our evaluation is restricted to TableQA scenarios—extending TORe to broader structured reasoning tasks (e.g., Text-to-SQL, knowledge graph QA) remains future work.

Future Directions. Promising extensions include: (1) dynamic operation pool expansion via automatic operation discovery; (2) lightweight recipe extraction using smaller models; (3) cross-modal recipe transfer between table, text, and knowledge graph domains.

References

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Yang Wang. 2020. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1026–1036.

Xiaochuang Chen and Kyunghyun Lee. 2024. Self-supervised representation learning for continual text classification. In *EMNLP*.

Yongrui Chen, Xinnan Guo, Tongtong Wu, Guilin Qi, Yang Li, and Yang Dong. 2023a. Learn from yesterday: A semi-supervised continual learning method for supervision-limited text-to-sql task streams. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 12682–12690.

Yongrui Chen, Shenyu Zhang, Guilin Qi, and Xinnan Guo. 2023b. Parameterizing context: unleashing the power of parameter-efficient fine-tuning and in-context tuning for continual table semantic parsing. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, pages 17795–17810.

Pablo García-Santaclara, Bruno Fernández-Castro, and Rebeca P Díaz-Redondo. 2025. Overcoming catastrophic forgetting in tabular data classification: A pseudorehearsal-based approach. *Engineering Applications of Artificial Intelligence*, 156:110908.

Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Mueller, Francesco Piccinno, and Julian Eisenschlos. 2020. Tapas: Weakly supervised table parsing via pre-training. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4320–4333.

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

Jianheng Huang, Leyang Cui, Ante Wang, Chengyi Yang, Xinting Liao, Linfeng Song, Junfeng Yao, and Jinsong Su. 2024. Mitigating catastrophic forgetting in large language models with self-synthesized rehearsal. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1416–1428, Bangkok, Thailand. Association for Computational Linguistics.

Hiroshi Iida, Dung Thai, Varun Manjunatha, and Mohit Iyyer. 2021. Tabbie: Pretrained representations of tabular data. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*.

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. *Preprint*, arXiv:2503.02240.

Zhuang Li, Lizhen Qu, and Reza Haffari. 2021. Total recall: a customized continual learning method for neural semantic parsers. In *Empirical Methods in Natural Language Processing 2021*, pages 3816–3831. Association for Computational Linguistics (ACL).

Ruiheng Liu, Jinyu Zhang, Yanqi Song, Yu Zhang, and Bailong Yang. 2025a. Discarding the crutches: Adaptive parameter-efficient expert meta-learning for continual semantic parsing. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3560–3578.

Ruiheng Liu, Jinyu Zhang, Yanqi Song, Yu Zhang, and Bailong Yang. 2025b. Filling memory gaps: Enhancing continual semantic parsing via sql syntax variance-guided llms without real data replay. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 24641–24649.

Fuli Qiao and Mehrdad Mahdavi. 2024. Learn more, but bother less: parameter efficient continual learning. In *The Thirty-eighth Annual Conference on Neural Information Processing Systems*.

Chengwei Qin and Shafiq Joty. 2022. LFPT5: A unified framework for lifelong few-shot language learning based on prompt tuning of t5. In *International Conference on Learning Representations*.

Bhaskarjit Sarmah, Dhagash Mehta, Benika Hall, Rohan Rao, Sunil Patel, and Stefano Pasquali. 2024. Hybridrag: Integrating knowledge graphs and vector retrieval augmented generation for efficient information extraction. In *Proceedings of the 5th ACM International Conference on AI in Finance*, pages 608–616.

Thomas Scialom, Tuhin Chakrabarty, and Smaranda Muresan. 2022. Fine-tuned language models are continual learners. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, pages 6107–6122.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. 2020. On the potential of lexico-logical alignments for semantic parsing to sql queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 1849–1864.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven CH Hoi. 2021. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation. In *EMNLP 2021-2021 Conference on Empirical Methods in Natural Language Processing, Proceedings*, pages 8696–8708. Association for Computational Linguistics (ACL).

Zhilin Wang, Yafu Li, Xiaoye Qu, and Yu Cheng. 2025. See: Continual fine-tuning with sequential ensemble of experts. *Preprint*, arXiv:2504.06664.

Zifeng Wang, Zizhao Zhang, Chen-Yu Lee, Han Zhang, Ruoxi Sun, Xiaoqi Ren, Guolong Su, Vincent Perot, Jennifer Dy, and Tomas Pfister. 2024a. Memory-based prompt augmentation for continual learning. In *NeurIPS*.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. 2024b. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*.

Jian Wu, Linyi Yang, Dongyuan Li, Yuliang Ji, Manabu Okumura, and Yue Zhang. 2025a. Mmqa: Evaluating llms with multi-table multi-hop complex questions. In *The Thirteenth International Conference on Learning Representations*, page 1.

Xianjie Wu, Jian Yang, Linzheng Chai, Ge Zhang, Jiaheng Liu, Xeron Du, Di Liang, Daixin Shu, Xianfu Cheng, Tianzhen Sun, and 1 others. 2025b. Tablebench: A comprehensive and complex benchmark for table question answering. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pages 25497–25506.

Dani Yogatama, Cyprien de Masson d'Autume, Jerome Connor, Tomas Kocisky, Mike Chrzanowski, Lingpeng Kong, Angeliki Lazaridou, Wang Ling, Lei Yu, Chris Dyer, and Phil Blunsom. 2019. Learning and evaluating general linguistic intelligence. *Preprint*, arXiv:1901.11373.

Jaehong Yoon, Saehoon Kim, Eunho Yang, and Sung Ju Hwang. 2020. Scalable and order-robust continual learning with additive parameter decomposition. In *International Conference on Learning Representations (ICLR)*.

Weixiang Zhao, Shilong Wang, Yulin Hu, Yanyan Zhao, Bing Qin, Xuanyu Zhang, Qing Yang, Dongliang Xu, and Wanxiang Che. 2024. Sapt: A shared attention framework for parameter-efficient continual learning of large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11641–11661.

Da-Wei Zhou, Zi-Wen Cai, Han-Jia Ye, Lijun Zhang, and De-Chuan Zhan. 2025. Dual consolidation for pre-trained model-based domain-incremental learning. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 20547–20557.

Jun-Peng Zhu, Peng Cai, Kai Xu, Li Li, Yishen Sun, Shuai Zhou, Haihuang Su, Liu Tang, and Qi Liu. 2024a. Autotqa: Towards autonomous tabular question answering through multi-agent large language models. *Proceedings of the VLDB Endowment*, 17(12):3920–3933.

Zhen Zhu, Yiming Gong, and Derek Hoiem. 2024b. Anytime continual learning for open vocabulary classification. In *European Conference on Computer Vision*, pages 269–285. Springer.

A Reproducibility Statement

To facilitate reproducibility, we commit to:

- **Code Release:** Full implementation (including data preprocessing, TOR extraction, and training scripts) will be released upon acceptance at `https://github.com/anonymous/TORe`.
- **Data Release:** Our constructed benchmarks (Mixed-QA-Stream and Mixed-Struct-Stream) will be publicly available in Alpaca format.
- **Hyperparameters:** All hyperparameters are detailed in our training configurations. We use Qwen-2.5-1.5B-Instruct and DeepSeek-R1-Distill-Llama-8B as backbone models with LoRA (rank=32). Training employs learning rate 2e-4 with AdamW optimizer.
- **Computational Resources:** Experiments conducted on 2×NVIDIA A100 (80GB) GPUs. Recipe extraction uses Meta-Llama-3.1-70B-Instruct accelerated by vLLM.

B Method Details

B.1 TOR Construction Process

The TOR construction exclusively utilizes Meta-Llama-3-70B-Instruct. For the given data, TOR

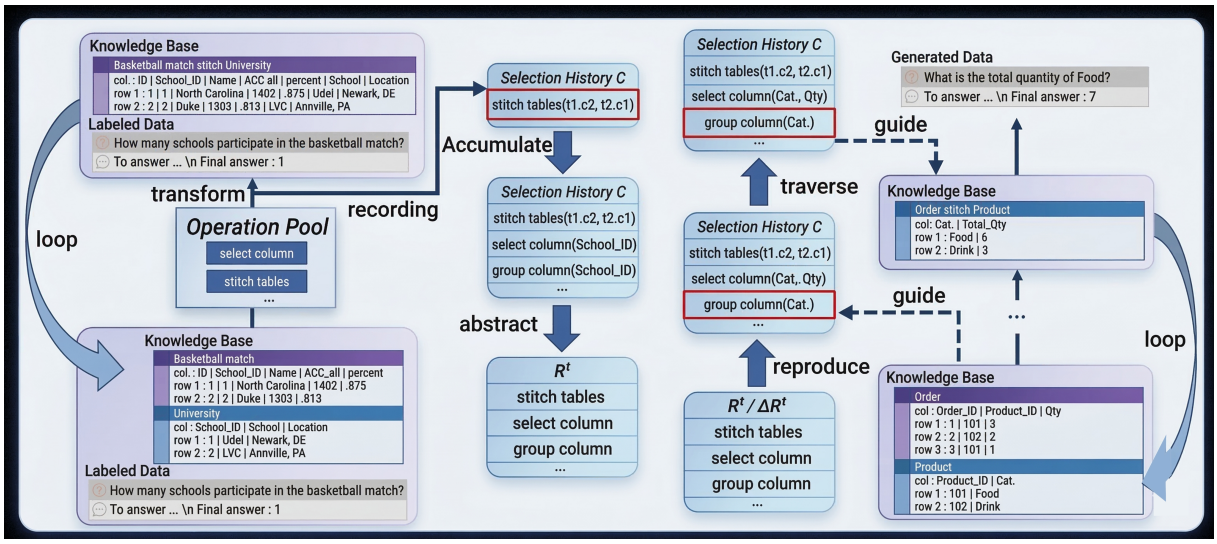


Figure 6: Detailed illustration of the Table Operation Recipe extraction process. Given a table and question, the LLM iteratively selects operations from the Operation Pool, generates content-specific parameters, and executes them to transform the table state. After the reasoning chain resolves the question, content-dependent parameters are stripped to form the abstract recipe.

processing follows the sequence of operation selection → output parameters → table processing → selection of next operation.

1. Before processing, it is first necessary to determine whether the current table to be processed is multi-table or single-table. If single-table, the table, question, and historical operations are directly input into the LLM, which provides the next operation. If multi-table, the LLM first selects either `select_table` or `stitch_tables` to convert the multi-table into a single table (as illustrated in Figure 10).

2. Due to the excessive number of available operations and the limited context provided by each operation, parameter selection is processed independently (as shown in Figure 12).

3. The operations and parameters are executed on the table according to the methods shown in Table 1, selecting predefined procedures to obtain an updated table.

B.2 Knowledge Difference Analysis

For each task, when computing the difference between the current task TOR and all past TORs, no clustering or deduplication is performed to strictly preserve the original distribution information.

B.3 TOR-Guided Data Synthesis

Algorithm 2 Consolidation Knowledge Construction

Input: Current task recipe set $\mathcal{R}^t$, Current task table distribution $\mathcal{D}_{\text{current}}$
Output: Consolidation Knowledge dataset $\mathcal{S}_{\text{CK}}$

Initialize $\mathcal{S}_{\text{CK}} \leftarrow \emptyset$
for each recipe $r \in \mathcal{R}^t$ **do**
  Sample table $T \sim \mathcal{D}_{\text{current}}$
  Instantiate parameter $\lambda \leftarrow \text{LLM_Gen}(r, T)$

  Executable sequence $\tilde{r} \leftarrow (r, \lambda)$
  $T' \leftarrow \text{Execute}(\tilde{r}, T)$
  $QA \leftarrow \text{LLM_Synthesize}(T, \tilde{r}, T')$
  $CoT \leftarrow \text{Format_Reasoning}(QA, \tilde{r}, T)$
  $\mathcal{S}_{\text{CK}} \leftarrow \mathcal{S}_{\text{CK}} \cup \{(QA, CoT)\}$
end for
return $\mathcal{S}_{\text{CK}}$

The extracted TORs are used for data synthesis. Current task data is combined with past and current TORs respectively and input to LLMs.

B.3.1 Data Preprocessing

While Mixed-QA-Stream features single-table data, Mixed-Struct-Stream is heterogeneous. If a target recipe specifies `select_table` but the sampled instance is a single table, the execution fails. To address this, we synthesize single-table inputs by randomly sampling from pure multi-tables, and conversely synthesize related multi-table contexts

using LLMs on pure single-tables.

B.3.2 Data Construction

We iterate through the constructed TORs, select a table from the current task, and prompt the LLM to perform parameter filling adapted to the current table (Figure 13 and Figure 14). The TOR and parameters are executed step-by-step. Since some operations cannot guarantee parameter filling quality purely from abstract recipes (e.g., concrete values from a table), we regenerate parameters dynamically (Figure 16). Finally, the original table, modified table, and TOR are used to synthesize complete QA pairs (Figure 15).

C Evaluation Metrics

Following prior work (Chen et al., 2023a; Li et al., 2021), we employ four standard continual learning metrics. Let $a_{j,i}$ denote accuracy on task i after training on task j, M the total number of tasks, and $\hat{a}_i$ the zero-shot accuracy on task i before any training:

Average Accuracy (ACC_a): The arithmetic mean of final accuracies across all tasks: $\text{ACC}_\text{a} = \frac{1}{M}\sum_{i=1}^{M} a_{M,i}$

Whole Accuracy (ACC_w): Accuracy evaluated on the combined test set of all tasks: $\text{ACC}_\text{w} = a_{\mathcal{D}_\text{test}^{(1:M)}}$

Forgetting Measure (FM): Quantifies catastrophic forgetting by measuring the performance degradation from the peak accuracy on each past task. $\text{FM} \geq 0$ by definition, and lower values indicate better retention: $\text{FM} = \frac{1}{M-1}\sum_{i=1}^{M-1}\left(\max_{j\in[i,M]} a_{j,i} - a_{M,i}\right)$

Forward Transfer (FWT): Examines zero-shot generalization to future tasks: $\text{FWT} = \frac{1}{M-1}\sum_{i=2}^{M}(a_{i-1,i} - \hat{a}_i)$

D Detailed Baseline Descriptions

To comprehensively evaluate TORe, we compare it against representative state-of-the-art methods across four categories.

1. Original Methods: SFT (Yogatama et al., 2019) serves as the lower bound, where the vanilla language model is fine-tuned sequentially on the task stream without any continual learning interventions, typically suffering from severe catastrophic forgetting.

2. Replay-based Methods: These methods explicitly store and replay past data. **RandSel** (Scialom et al., 2022) constructs a memory buffer by randomly sampling 10% of data from previous tasks. **KMeansSel** (Wang et al., 2025) improves upon this by using K-Means clustering to select the most representative 10% of historical data. **MTL** (Chen et al., 2023a) represents the empirical upper bound for rehearsal methods by replaying 100% of the historical data alongside the current task.

3. PEFT Methods: C3 (Continual Context Customization) (Chen et al., 2023b) utilizes Parameter-Efficient Fine-Tuning dynamically but operates under an idealized assumption: it requires gold-truth task ID labels during inference to select the correct parameter routing. **SEE** (Wang et al., 2025) represents a hybrid approach, integrating data recall (replay) with PEFT methodologies to preserve early representations while accommodating new task features.

4. Data Synthesis Methods: These are strictly rehearsal-free approaches. **Naive Generate** relies entirely on the LLM's intrinsic knowledge to synthesize generic table QA pairs via zero-shot/few-shot prompting, lacking structural constraints. **SSR** (Synthetic Sample Refinement) (Huang et al., 2024) synthesizes pseudo-samples based on the current context and refines them using the latest model weights. We evaluate SSR at different generation scales (1%, 10%, and matched volume).

E Qualitative Analysis

E.1 Case Study: Knowledge Transfer in Action

Consider transitioning from Task 1 (Financial databases requiring multi-table `stitch_tables` and `group_by`) to Task 3 (Medical single-tables requiring simple `select_row`). When training on Task 3, $\Delta\mathcal{R}^t$ identifies the missing `group_by` recipe. TORe samples a medical table containing "Patient ID" and "Treatment Costs", and the LLM projects the `group_by` operation onto this schema, synthesizing: *"What is the total cost grouped by Treatment Type?"* This effectively injects the historical "aggregation" reasoning skill into the current domain, maintaining Task 1 proficiency while strictly preserving financial data privacy.

E.2 Error Analysis

To understand TORe's limitations, we manually analyzed 100 failure cases across both benchmarks. We identify four primary error categories:

(1) Complex Multi-Table Reasoning (38%): When historical recipes involve deeply nested op-

erations (e.g., 3-way table joins + nested aggregation), projecting these onto structurally simpler current tables causes LLM hallucination. While execution-guided verification rejects invalid syntheses, it results in high rejection rates (up to 62% for recipes with >5 operations).

(2) Numerical Reasoning Errors (31%): LLMs occasionally generate parameters with incorrect numerical constraints (e.g., `sort_column(age, threshold=200)` for human age data). Although executable, these produce semantically invalid questions, degrading training quality.

(3) Domain-Specific Knowledge Gaps (21%): For highly specialized domains (e.g., medical terminology, chemical formulas), the LLM lacks domain expertise to generate realistic QA pairs, leading to shallow or incorrect questions.

(4) Others (10%): Miscellaneous errors including formatting issues and edge cases.

These findings suggest future directions: (1) incorporating domain-specific knowledge bases; (2) enhanced semantic validation beyond execution correctness; (3) adaptive complexity-aware recipe selection.

F Detailed Dataset Construction

TableInstruct (Wu et al., 2025b): A comprehensive dataset containing 3,681 single tables spanning 20 domains, originally designed to enhance LLM instruction following on tables. We utilize its diverse domains to construct distinct sequential tasks.

SQUALL (Shi et al., 2020): An extension of WikiTableQuestions (WTQ) featuring manually paraphrased natural language questions aligned with SQL equivalents, focusing on complex column/row filtering.

MMQA (Wu et al., 2025a): A multimodal dataset featuring complex multi-hop reasoning over tables, text, and images sourced from Wikipedia. We extract its multi-table sub-graphs to challenge the model's schema-linking capabilities in our Mixed-Struct-Stream.

OMNISQL (Li et al., 2025): A large-scale Text-to-SQL benchmark containing complex multi-table join operations across varied database schemas. It serves as a rigorous testbed for evaluating the transition from single-table to deeply nested multi-table reasoning in our task stream.

G Operation Pool Coverage Analysis

To validate the sufficiency of our 9 predefined operations, we analyze their coverage on 500 randomly sampled TableQA instances across all datasets.

Task Type	Samp.	Full	Part.	None	%Cov.
Single (SQUALL)	150	149	1	0	99.3
Single (TableInst.)	150	148	2	0	98.7
Multi (MMQA)	100	85	13	2	85.0
Multi (OMNISQL)	100	90	8	2	90.0
Total	**500**	**472**	**24**	**4**	**94.3**

Table 5: Operation pool coverage by task type. "Full" means all reasoning steps are expressible using our 9 operations. The 9 uncovered cases involve domain-specific computations (e.g., financial IRR, statistical hypothesis testing) that rarely appear in general TableQA.

H Recipe Extraction Quality Analysis

We validate the quality of LLM-generated Table Operation Recipes on 200 randomly sampled instances across both benchmarks. Three annotators independently assess whether each recipe: (1) correctly captures the reasoning chain, (2) uses appropriate operations, and (3) maintains parameter-operation consistency (inter-annotator Fleiss' $\kappa = 0.87$).

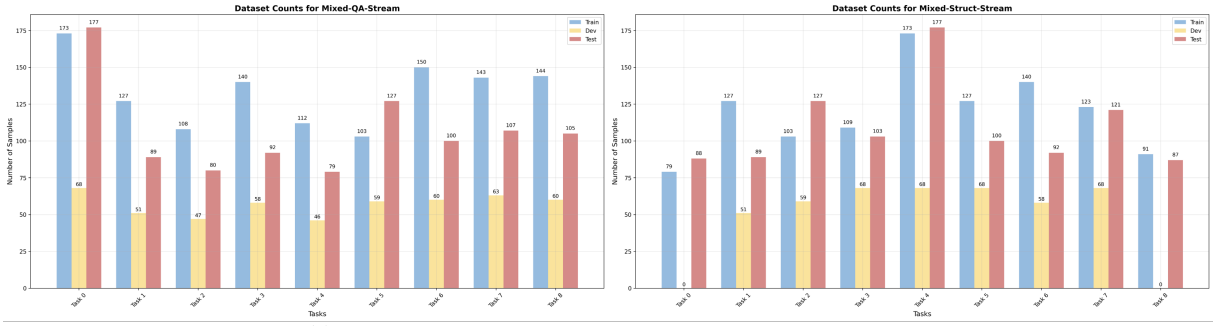
Quality Level	Pre-Verif.	Post-Verif.
Fully correct	179 (89.5%)	198 (99.0%)
Minor errors	14 (7.2%)	1 (0.5%)
Major errors	7 (3.3%)	1 (0.5%)
Usable rate	**96.7%**	**99.5%**

Table 6: Recipe extraction quality on 200 manually verified samples. Execution-guided verification reduces the major error rate from 3.3% to 0.5%.

The 7 pre-verification major errors stem from ambiguous questions (57%), complex multi-table joins (29%), and underrepresented operations (14%). Execution-guided verification catches 6 out of 7 major errors (85.7% catch rate), demonstrating its effectiveness as a quality control mechanism. This two-stage approach (LLM generation + execution verification) ensures high-quality recipe extraction that forms a reliable foundation for downstream knowledge reconstruction.

I Theoretical Justification

We provide theoretical insights into why recipe-based abstraction enables effective continual learn-



(a) (b)

Figure 7: The specific data volume for each task in Mixed-QA-Stream and Mixed-Struct-Stream

ing.

Information Bottleneck Perspective. Traditional methods learn $P(Y|X)$ directly from table content X, overfitting to task-specific semantics. TORe introduces recipes $R = \phi(X)$ and learns $P(Y|R)$ instead. Following the Information Bottleneck principle, recipes minimize $I(X; R)$ (discarding schema-specific details) while maximizing $I(R; Y)$ (preserving reasoning patterns), forcing the model to learn generalizable structural patterns that transfer across schema changes.

Schema Invariance. Let T_1, T_2 be tables with different schemas but identical structural reasoning requirements. Since recipe extraction removes all content-dependent parameters λ, retaining only schema-agnostic operations (Table 1), the extracted recipes $\phi(T_1, Q_1) = \phi(T_2, Q_2)$ whenever Q_1 and Q_2 require the same reasoning steps. This invariance explains TORe's 99.24% Order Stability: learning is fundamentally decoupled from task-specific surface features.

Generalization Bound. TORe's hypothesis space is constrained by the operation pool size ($|\mathcal{O}| = 9$) rather than the vocabulary size ($|\mathcal{V}|$). The effective VC dimension is $O(\log |\mathcal{O}|) \ll O(\log |\mathcal{V}|)$, providing a tighter generalization bound. This theoretical advantage translates to TORe's empirically lower FM (1.03 vs. SSR's 10.91) and higher accuracy.

J More Results and Analysis

J.1 Results with Meta-Llama-3.2-1B-Instruct

To further validate TORe's generalizability across different model backbones, we conduct experiments using Meta-Llama-3.2-1B-Instruct. As shown in Table 7, TORe consistently achieves the best performance among all methods on both benchmarks.

On this smaller backbone, TORe achieves 19.40% ACC_a on Mixed-QA-Stream (+3.31 over the best baseline SEE) and 24.04% on Mixed-Struct-Stream (+0.53 over MTL). The consistent improvement across Qwen-2.5-1.5B (Table 2), DeepSeek-R1-8B (Table 2), and Meta-Llama-3.2-1B confirms that TORe's recipe-based reconstruction is model-agnostic and robust across different backbone architectures and scales.

J.2 Analysis of Forgetting Measure (FM)

As shown in Table 2, the FM results in our TableQA-CL scenario exhibit interesting patterns worthy of deeper analysis. Unlike traditional continual learning scenarios where forgetting measures typically show severe degradation (large positive values indicating substantial forgetting), our TableQA-CL setting demonstrates relatively controlled forgetting across most methods.

Understanding FM in TableQA-CL Context. The FM metric measures the gap between peak performance on each task (which may occur at any point from that task's training through all subsequent tasks) and the final performance after training on all tasks. Formally, for task i evaluated after completing task M:

$$\text{FM}_i = \max_{j \in [i,M]} a_{j,i} - a_{M,i} \tag{4}$$

where $a_{j,i}$ denotes the accuracy on task i after training on task j. A lower FM indicates better knowledge retention.

Why TableQA-CL Shows Moderate FM Values. Due to the special nature of the TableQA-CL scenario, where *abstract reasoning patterns* (e.g., row selection, column grouping, table joining) are

Method	Type	Mixed-QA-Stream				Mixed-Struct-Stream			
		ACC_a	ACC_w	FM	FWT	ACC_a	ACC_w	FM	FWT
SFT	ORIGINAL	14.85	16.00	**0.89**	8.31	19.75	20.33	1.81	12.86
MTL	RECALL	11.01	12.65	1.31	6.04	23.51	23.17	1.79	**20.55**
SEE	RECALL&PEFT	16.09	17.05	1.18	10.73	23.11	23.68	**1.57**	15.45
C3♢	PEFT	14.40	15.05	–	–	15.72	15.96	–	–
SSR(1%)	SYNTHESIS	9.75	9.88	7.44	7.77	16.86	17.58	3.52	12.08
SSR(10%)	SYNTHESIS	10.71	10.50	5.61	6.91	17.37	18.29	2.37	12.74
SSR	SYNTHESIS	10.55	10.71	5.93	6.70	9.35	9.55	7.53	4.85
TORe(only CK)	SYNTHESIS	15.08	15.96	3.16	10.86	22.60	24.05	4.07	16.09
TORe(only HK)	SYNTHESIS	15.66	16.84	2.96	11.53	22.77	24.09	4.66	17.21
TORe	SYNTHESIS	**19.40**	**20.69**	2.27	**14.12**	**24.04**	**24.90**	3.12	18.60

Table 7: Results on Meta-Llama-3.2-1B-Instruct. TORe achieves the highest ACC_a on both benchmarks (+3.31 over SEE on Mixed-QA-Stream, +0.53 over MTL on Mixed-Struct-Stream), demonstrating consistent effectiveness across different model backbones.

shared across diverse table schemas, the FM values exhibit several distinctive characteristics:

(1) Positive Transfer Effects: Unlike image classification or text generation tasks where task distributions are highly disjoint, TableQA tasks share common structural reasoning operations. Therefore, training on later tasks can actually *improve* performance on earlier tasks through enhanced abstract reasoning capabilities. This manifests as the peak accuracy $\max_{j\in[i,M]} a_{j,i}$ often occurring *after* task i itself, naturally bounding FM from below.

(2) Schema-Agnostic Skills: Operations like `select_row`, `group_column`, and `sort_column` are domain-invariant. A model trained on financial tables (Task 1) that learns to perform aggregation operations can readily apply this skill to medical tables (Task 3), mitigating catastrophic forgetting at the reasoning level even if semantic vocabulary differs entirely.

(3) Low Data Regime Amplifies Generalization: With fewer than 200 training samples per task, models cannot overfit to task-specific surface patterns. Instead, they are forced to learn generalizable structural patterns, which naturally transfer across tasks and reduce forgetting.

Comparative Analysis Across Methods. Despite these favorable conditions, substantial differences exist between methods:

Rehearsal-based methods (RandSel, KMeansSel, MTL, SEE) achieve moderate FM values (1.26-5.89) by directly replaying historical data or leveraging PEFT mechanisms. SEE achieves the best FM among this category (1.26 on Mixed-Struct-Stream) by combining data replay with adaptive PEFT parameter selection.

Synthesis-based methods without structural control (SSR, Naive generate) suffer from dramatically higher FM (6.22-10.91). SSR's FM of 10.91 on Mixed-QA-Stream indicates catastrophic forgetting despite synthetic data augmentation, revealing that unverified LLM-generated data introduces noise that actively *harms* knowledge retention. The more synthetic data SSR uses, the worse its FM becomes (compare SSR(2%): 8.33, SSR(20%): 9.24, SSR: 10.91), validating that quality trumps quantity.

TORe's superior FM performance (0.81-1.03 on Mixed-QA-Stream, 1.85-1.99 on Mixed-Struct-Stream) stems from three key mechanisms:

- **Explicit Deviation Modeling:** By computing $\Delta\mathcal{R}^t$, TORe precisely identifies which historical reasoning patterns are absent from the current task and reconstructs them, directly addressing the root cause of forgetting.
- **Execution-Guided Verification:** Only successfully executable synthetic examples are retained, ensuring high-quality data that reinforces correct reasoning patterns rather than introducing hallucinated noise.
- **Balanced CK-HK Synthesis:** The 1HK:3CK ratio ensures sufficient current-task learning (via CK) while maintaining historical proficiency (via HK), achieving an optimal forgetting-acquisition trade-off.

The Trade-off Between Forgetting and Learning. An intriguing observation from Table 4 is that vanilla SFT ($N_K = 0$) achieves the lowest FM

of 0.80, yet its ACC_a of only 23.50 is far below TORe's 27.51. This reveals a fundamental tension: *aggressive knowledge acquisition inevitably incurs some forgetting*. Methods that avoid forgetting entirely (FM ≈ 0) typically fail to learn new tasks effectively, while methods that learn aggressively (high ACC_a) risk catastrophic forgetting (high FM). TORe's achievement of **simultaneously high ACC_a and low FM** demonstrates its ability to navigate this trade-off via structurally-controlled knowledge reconstruction.

Implications for Future Work. The moderate FM values in TableQA-CL suggest that structural reasoning tasks may be inherently more amenable to continual learning than previously studied domains. However, this does *not* diminish the importance of anti-forgetting mechanisms—as evidenced by SSR's catastrophic failure (FM = 10.91) without proper structural control. Future work should explore: (1) whether similar recipe-based abstractions can be derived for other structured reasoning tasks (e.g., program synthesis, mathematical reasoning); (2) how to automatically discover optimal CK-HK ratios per task; (3) techniques to further reduce FM while maintaining high ACC_a in scenarios with extreme schema heterogeneity.

J.3 Impact of Synthesized Data Amount

We provide detailed results on the impact of different amounts of synthesized data on various metrics (Figure 8). As the amount of synthesized data scales up to $N_K = 400$, both ACC_a and FWT reach their optimal maximum values, while FM achieves its optimal *minimum* value of approximately 1.0, indicating minimal forgetting. This sweet spot represents the ideal balance where the model acquires sufficient new knowledge without introducing excessive synthetic noise.

When the data amount increases further ($N_K > 400$), performance becomes unstable: ACC_a begins to decline, FM increases (indicating more forgetting), and FWT fluctuates. This degradation proves that **massive hallucinated data without structural control actively harms the model**. Specifically, at $N_K = 1200$, FM rises to 3.46 (more than 3× the optimal value), revealing that over-synthesized low-quality examples confuse the model's internal representations, causing it to forget previously learned reasoning patterns.

The U-shaped FM curve validates our core hypothesis: *structural fidelity matters more than sheer quantity*. TORe's execution-guided verification ensures that each synthetic example faithfully represents a valid reasoning path, but when synthesis scales beyond the diversity of underlying Operation Recipes, redundant low-quality samples accumulate, degrading retention of historical knowledge.

K Prompt Examples and Data Formats

Below are visual representations of the prompts and data formats used during TOR extraction and dataset synthesis.

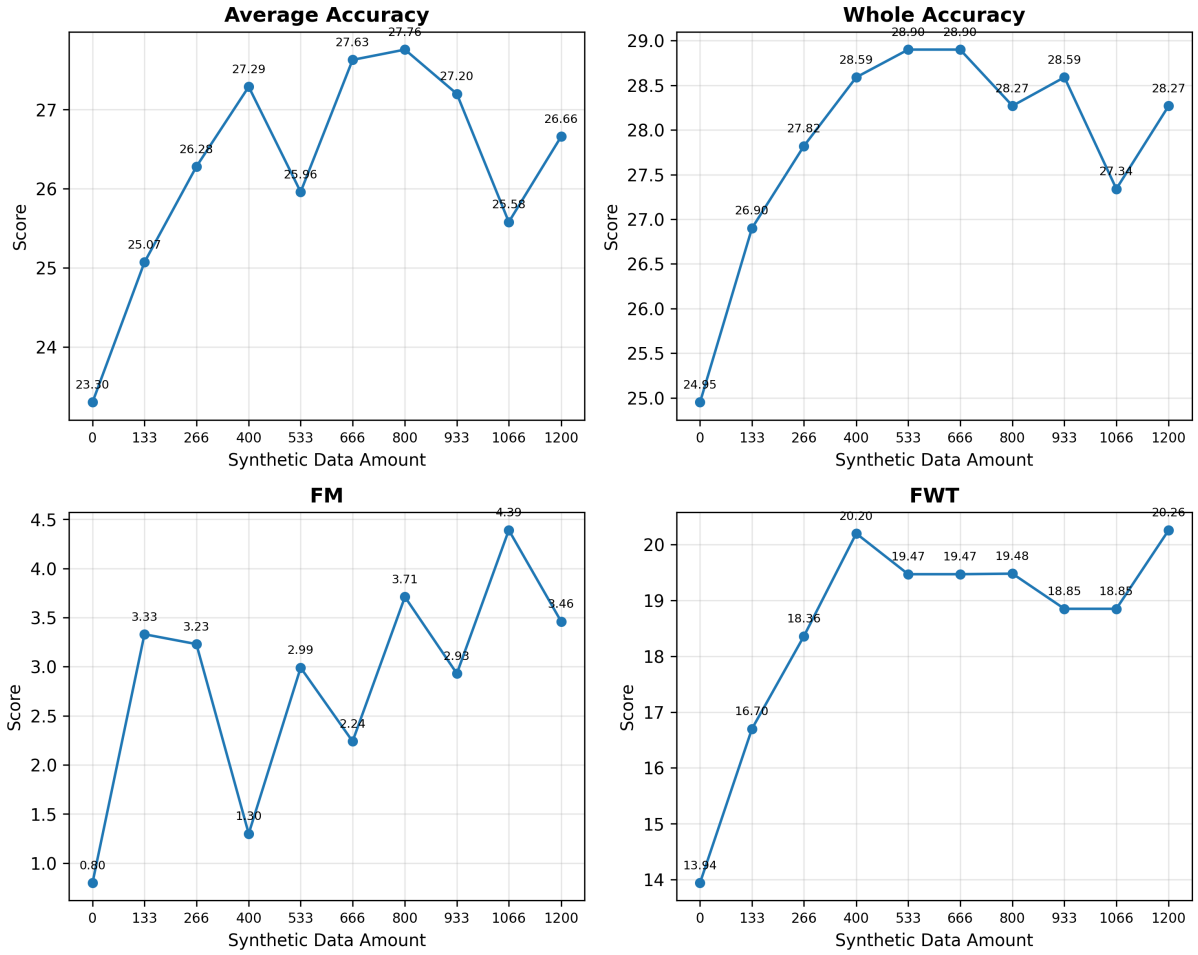


Figure 8: Impact of synthesized data amount on results

"ID": "203-339",
"Instruction": """You are a table analyst. Your task is to answer questions based on the table content.\n\n\nThe answer should follow the format below:\n[Answer Format]\nFinal Answer: AnswerName1, AnswerName2...\n\nEnsure the final answer format is the last output line and can only be in the \"Final Answer: AnswerName1, AnswerName2...\" form, no other form. Ensure the \"AnswerName\" is a number or entity name, as short as possible, without any explanation.\n\n\nLet's think step by step and then give the final answer to the question.\n\nRead the table below in JSON format:\n
##Instruction:{"columns": ["Year", "Car", "Start", "Qual", "Rank", "Finish", "Laps", "Led", "Retired"], "data": [["1961", "98", "5", "146.080", "7", "12", "192", "27", "Flagged"], ["1962", "98", "1", "150.370", "1", "7", "200", "120", "Running"], ["1963", "98", "1", "151.153", "1", "1", "200", "167", "Running"], ["1964", "98", "4", "155.099", "4", "23", "55", "7", "Pit fire"], ["1965", "98", "5", "158.625", "5", "2", "200", "0", "Running"], ["1966", "98", "4", "162.484", "4", "14", "87", "0", "Wheel Bearing"], ["1967", "40", "6", "166.075", "6", "6", "196", "171", "Bearing"], ["Totals", "Totals", "Totals", "Totals", "Totals", "Totals", "1130", "492", ""]]}""",
"Input": "###Input:\nwhat car achieved the highest qual?\n\n###Response:",
"Output": "To determine what car achieved the highest qual?, we need to write the sql corresponding to the question. The sql is ```sql\nselect c2 from w order by c4_number desc limit 1\n```\n\nFinal Answer: 40"

Figure 9: An example of the data format of the dataset we constructed

Instruction
You are a table processing assistant. Your task is to analyze a question and a set of tables and decide the first step to take.

Output Format
You MUST follow one of these exact formats:

1. If you need to combine information from two tables:
Explanation: [explanation of why you need to combine tables]
Function: f_stitch_tables(table_name1.column_name, table_name2.column_name)

2. If the answer can be found in a single table:
Explanation: [explanation of why only one table is needed]
Function: f_select_table(table_name)

Rules
1. You MUST choose exactly one function: `f_stitch_tables` or `f_select_table`.
2. For `f_stitch_tables`, you MUST specify the two tables and the columns to join on.
3. For `f_select_table`, you MUST specify the name of the table to keep.

Example 1 (Stitching is needed)
/*
{Example_1_table}
*/
Question: {Example_1_question}
Explanation: {Example_1_answer}
Function: f_stitch_tables({Example_1_table_1_name, Example_1_table_2_name)

Example 2 (Single table is sufficient)
/*
{Example_2_table}
*/
Question: {Example_2_question}
Explanation: {Example_2_answer}
Function: f_select_table({Example_2_table_?_name})

Current Tables and Question
/*
{user_table}
*/
Question: {user_question}

Figure 10: Prompt to prompting LLMs to generate next step (stitch_tables or select_table).

Instruction
You are a table processing assistant. Your task is to select the correct operation to process the given table step by step to answer the question.

Output Format
You MUST follow this exact format:

1. For Operation (except END):
Explanation: [explanation]
Function: [function]

2. For END:
Explanation: [explanation]
Function: END
Answer: [your final answer]

Rules
1. You MUST choose exactly one operation from the available operations
2. You MUST follow the exact output format
3. For table operations, you MUST include the complete new table
4. For END operation, you MUST provide a clear and concise answer
5. Do not include any explanations or additional text

Available Operations
{Operations not been skipped}

Examples
/*
{f_add_knowledge_column_example_table}
*/
Question: {f_add_knowledge_column_example_question}
Explanation: {f_add_knowledge_column_example_explanation} If the table does not have the needed column to answer the question and the column need extra knowledge, we use f_add_knowledge_column() to add a new column for it. We add a column "xxx".
Function: f_add_knowledge_column(xxx)

……

Current Tables and Question
/*
{user_table}
*/
Question: {user_question}
Explanation:

Figure 11: Prompt to prompting LLMs to generate next step (other operations except stitch_tables or select_table).

To answer the question, we can first use f_add_column() to add one column to the table.

The added column should have these data types:
1. Numerical: the numerical strings that can be used in sort, sum
2. Datetype: the strings that describe a date, such as year, month, day
3. String: other strings

Example 1
/*
{Example_1_table}
*/
Question: {Example_1_question}
The existing columns are: {Example_1_existing_columns}
Explanation: {Example_1_explanation}. The datatype is xxx.
Therefore, the answer is: f_add_column(xxx). The value: xxx
New Table:
/*
{Example_1_new_table}
*/

Example 2
/*
{Example_2_table}
*/
Question: {Example_2_question}
The existing columns are: {Example_2_existing_columns}
Explanation: {Example_2_explanation}. The datatype is xxx.
Therefore, the answer is: f_add_column(xxx). The value: xxx
New Table:
/*
{Example_2_new_table}
*/

Current Tables and Question
/*
{user_table}
*/
Question: {user_question}
The existing columns are: {user_existing_columns}
Explanation:

Figure 12: One of the prompts used in the parameter generation process.

You are a table operation parameter generation assistant. Your task is to fill in the parameters for each operation in the chain based on the given table.

For each operation in the chain, you need to:
1. Analyze the table structure and content
2. Determine appropriate parameters for each operation
3. Ensure the parameters are valid and meaningful
4. New intermediate table should be inferred from the last intermediate table

Available Operations and their parameter requirements:
- f_add_knowledge_column(column_name): Add a new column that requires external knowledge
- f_add_inferred_column(column_name): Add a new column that can be calculated or inferred from existing columns
- f_sort_column(column_name): Sort the table by a specific column
- f_select_column(column1, column2, ...): Select specific columns from the table
- f_select_row(row1, row2, ...): Select specific rows from the table
- f_group_column(column_name): Group the table by only one specific column, the output table should only have the specified column and the count column
- f_change_column_name(old_name, new_name): Rename a column
- f_stitch_tables(table1.column1, table2.column2, join_method): Stitch two tables together by a specific column, choose join_method in [inner, left, right]
- f_select_table(table_name): Select a specific table from the database

Please output in the following format:
Operation 1: [operation with filled parameters]
Explanation: [explain why these parameters were chosen]
Intermediate Table: [show the table after this operation]

Operation 2: [operation with filled parameters]
Explanation: [explain why these parameters were chosen]
Intermediate Table: [show the table after this operation]

Example 1:
Table Information:
{table_info_example_1}
Operation Chain:
{KBOR_example_1}
Filled Chain:
{filled_ KBOR _example_1}

Example 2:
Table Information:
{table_info_example_2}
Operation Chain:
{KBOR _example_2}
Filled Chain:
{filled_ KBOR _example_2}

Example 3:
Table Information:
{table_info_example_3}
Operation Chain:
{KBOR _example_3}
Filled Chain:
{filled_ KBOR _example_3}

Your Turn:
Table Information:
{table_info}
Operation Chain:
{KBOR}
Filled Chain:

Figure 13: Prompt for filling parameters in TOR given a table.

An example of parameter filling:

Table Information:
/*
table_name: basketball_match
col : Team_ID | School_ID | Team_Name | ACC_Regular_Season | ACC_Percent | ACC_Home | ACC_Road | All_Games | All_Games_Percent | All_Home | All_Road | All_Neutral
row 1 : 1 | 1 | North Carolina | 14â2 | .875 | 6â2 | 8â0 | 35â2 | 0.946 | 14â2 | 13â0 | 9â1
row 2 : 2 | 2 | Duke | 13â3 | .813 | 7â1 | 6â2 | 28â6 | 0.824 | 15â1 | 8â2 | 5â3
row 3 : 3 | 4 | Clemson | 10â6 | .625 | 7â1 | 3â5 | 24â10 | 0.706 | 14â2 | 6â5 | 4â3
row 4 : 4 | 5 | Virginia Tech | 9â7 | .563 | 6â2 | 3â5 | 21â14 | 0.6 | 14â3 | 4â8 | 3â3
foreign_key: journal_id, editor_id
*/

Operation Chain:
["f_sort_column()", "f_select_row()", "f_select_column()", "END"]

Filled Chain:
Operation 1: f_sort_column(All_Games_Percent)
Explanation: Sorting by the All_Games_Percent column in descending order to find the school with the highest percentage.
Intermediate Table:
/*
col : Team_ID | School_ID | Team_Name | ACC_Regular_Season | ACC_Percent | ACC_Home | ACC_Road | All_Games | All_Games_Percent | All_Home | All_Road | All_Neutral
row 1 : 1 | 1 | North Carolina | 14â2 | .875 | 6â2 | 8â0 | 35â2 | 0.946 | 14â2 | 13â0 | 9â1
row 2 : 2 | 2 | Duke | 13â3 | .813 | 7â1 | 6â2 | 28â6 | 0.824 | 15â1 | 8â2 | 5â3
row 3 : 3 | 4 | Clemson | 10â6 | .625 | 7â1 | 3â5 | 24â10 | 0.706 | 14â2 | 6â5 | 4â3
row 4 : 4 | 5 | Virginia Tech | 9â7 | .563 | 6â2 | 3â5 | 21â14 | 0.6 | 14â3 | 4â8 | 3â3
*/

Operation 2: f_select_row(row 1)
Explanation: Selecting the first row which corresponds to the school with the highest All_Games_Percent.
Intermediate Table:
/*
col : Team_ID | School_ID | Team_Name | ACC_Regular_Season | ACC_Percent | ACC_Home | ACC_Road | All_Games | All_Games_Percent | All_Home | All_Road | All_Neutral
row 1 : 1 | 1 | North Carolina | 14â2 | .875 | 6â2 | 8â0 | 35â2 | 0.946 | 14â2 | 13â0 | 9â1
*/

Operation 3: f_select_column(Team_Name, ACC_Regular_Season)
Explanation: Selecting only the Team_Name and ACC_Regular_Season columns for the result.
Intermediate Table:
/*
col : Team_Name | ACC_Regular_Season
row 1 : North Carolina | 14â2
*/

Operation 4: END

Figure 14: An example of parameter filling.

You are a table question generation assistant. Your task is to generate a natural question and its corresponding answer based on the table processing steps.

Please generate a question and answer that:
1. Requires all the operations in the chain to solve
2. Can be answered using the final table
3. Is natural and clear
4. Has a precise answer

Please output in the following format:
Question: [generated question]
Answer: [generated answer]
Explanation: [explain how the operations help answer this question]

Example 1:
Original Table:
{original_table_example_1}
Operation Chain with Parameters:
{chain_with_params_example_1}
Final Table:
{final_table_example_1}
Question:
{question_example_1}
Answer:
{answer_example_1}
Explanation:
{explanation_example_1}

Example 2:
Original Table:
{original_table_example_2}
Operation Chain with Parameters:
{chain_with_params_example_2}
Final Table:
{final_table_example_2}
Question:
{question_example_2}
Answer:
{answer_example_2}
Explanation:
{explanation_example_2}

Your Turn:
Original Table:
{original_table}
Operation Chain with Parameters:
{chain_with_params}
Final Table:
{final_table}
Question:

Figure 15: Prompt of question-answer pairs generation

We use f_add_inferred_column() to add one column to the table. This function is used when we need to add a column that can be calculated or inferred from existing columns. You need to add the values of the column.

Response format
Return the values in this format:
[value1, value2, ...]

Example 1
/*
{Example_1_table}
*/
Function: f_add_inferred_column(xxx)
Explanation: {Example_1_explanation}
Values: [xxx]
New Table:
/*
{Example_1_new_table}
*/

Example 2
/*
{Example_2_table}
*/
Function: f_add_inferred_column(xxx)
Explanation: {Example_2_explanation}
Values: [xxx]
New Table:
/*
{Example_2_new_table}
*/

Your Turn
[user_table]
Function: [user_function]
Explanation: [user_explanation]
Values:

Figure 16: Prompt of refilling the values of the operation