Adaptive and Robust Translation from Natural Language to Multi-model Query Languages

Gengyuan Shi, Chaokun Wang, Yabin Liu, Jiawei Ren
School of Software, BNRist, Tsinghua University, Beijing 100084, China
shigy19@mails.tsinghua.edu.cn,chaokun@tsinghua.edu.cn,
liu-yb23@mails.tsinghua.edu.cn,rjw0238@gmail.com

Abstract

Multi-model databases and polystore systems are increasingly studied for managing multi-model data holistically. As their primary interface, multi-model query languages (MMQLs) often exhibit complex grammars, highlighting the need for effective Text-to-MMQL translation methods. Despite advances in natural language translation, no effective solutions for Text-to-MMQL exist. To address this gap, we formally define the Text-to-MMQL task and present the first Text-to-MMQL dataset involving three representative MMQLs. We propose an adaptive Text-to-MMQL framework that includes both a schema embedding module for capturing multi-model schema information and an MMQL representation strategy to generate concise intermediate query formats with error correction in generated queries. Experimental results show that the proposed framework achieves over a 9% accuracy improvement over our adapted baseline methods.

1 Introduction

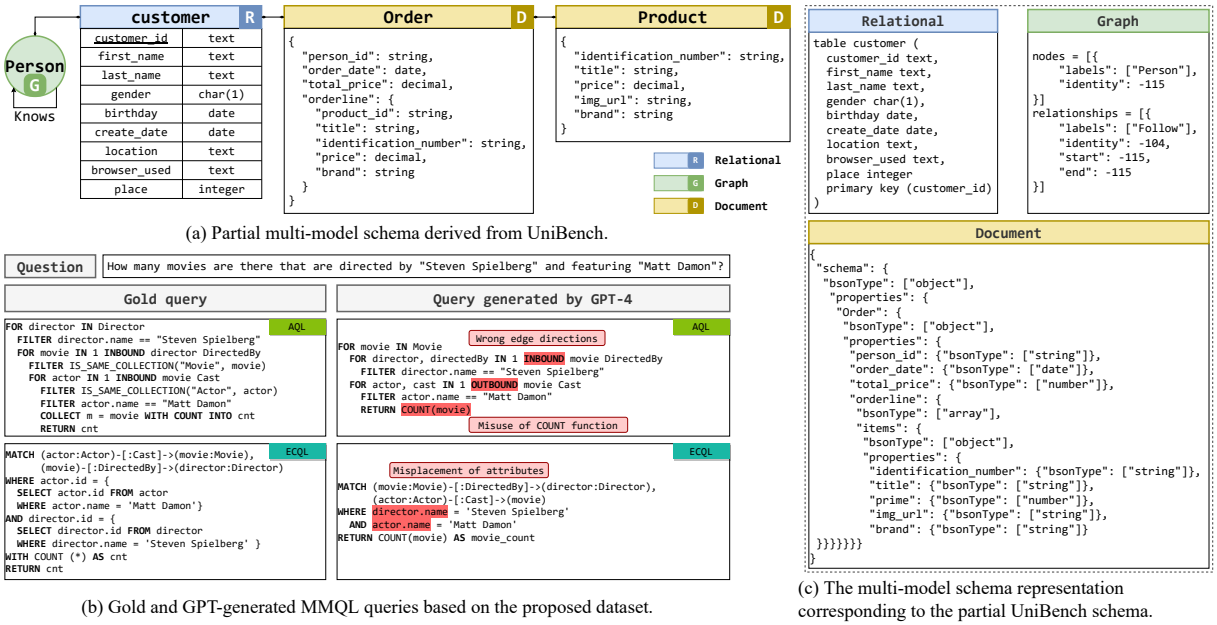
With the development of diverse data management technologies, modern applications increasingly rely on multi-model data or multiple types of databases simultaneously. On top of this tendency, multi-model databases (a.k.a. polystores) have been proposed and widely studied according to the "No one size fits all" principle (Stonebraker and Çetintemel, 2005). These systems provide multi-model query languages (MMQLs) as unified interfaces to access and analyze multi-model data, enabling comprehensive insights and enhancing decision-making processes in many modern data management applications (Ooi et al., 2023; Shi et al., 2024b).

As an essential task in natural language processing studies (Li et al., 2023a), natural language to query generation becomes especially challenging in multi-model data management scenarios, due to the lack of a standardized MMQL and the significant grammatical diversity across existing MMQLs (Zhang et al., 2018). While recent advancements in Text-to-SQL translation show promise (Li et al., 2023a,b; Wang et al., 2020; Scholak et al., 2021; Hui et al., 2022; Fu et al., 2023), these methods are not directly applicable to MMQLs due to grammatical discrepancies. The Text-to-MMQL task faces inherent challenges, including the lack of datasets and the complexity of MMQLs, as illustrated in Figures 1a and 1b based on the partial data schema of the UniBench (Zhang et al., 2018; Zhang and Lu, 2021) benchmark for multi-model databases, with MMQLs including AQL (ara, 2025) and ECQL (Shi et al., 2024a).

Lack of datasets. To the best of our knowledge, no publicly available dataset currently exists for the Text-to-MMQL task, preventing the application of data-driven learning approaches. While large language models (LLMs) have demonstrated strong performance on Text-to-SQL generation (Liu et al., 2023; Dong et al., 2023; Ren et al., 2024), their parametric knowledge constraints lead to performance degradation when handling MMQLs outside their training corpus. For example, Figure 1b shows GPT-4 (Achiam et al., 2023) generates a Cypher query instead of the expected ECQL query with nested SQL sub-queries.

Complexity of multi-model databases and MMQLs. The Text-to-MMQL task presents three fundamental challenges: (1) *Schema Heterogeneity*: While multi-model schema information improve translation accuracy (Li et al., 2023a,b; Ren et al., 2024), their unified representation remains problematic (Stonebraker and Çetintemel, 2005). Current relational schema-focused methods cannot adequately capture multi-model schema information such as graph edges and nested document structures, as shown in Figure 1a. (2) *Language Heterogeneity*: Existing MMQLs' grammatical diversity invalidates SQL-specific techniques based on sketches (Li et al., 2023a), intermediate representation (Gan et al., 2021; Fu et al., 2023),

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 15950–15965



(a) Partial multi-model schema derived from UniBench.

(b) Gold and GPT-generated MMQL queries based on the proposed dataset.

(c) The multi-model schema representation corresponding to the partial UniBench schema.

Figure 1: Illustration of multi-model schema and the Text-to-MMQL translation task.

and rule-based post-processing (Fu et al., 2023). LLM-based methods also face performance degradation on unfamiliar MMQLs. As shown in Figure 1b, GPT-4 incorrectly uses SQL-like `COUNT()` in AQL, where the correct syntax requires "`COLLECT movie WITH COUNT`". (3) *Language Ambiguity*: Loose constraints in multi-model databases and polystores complicate error detection in the generated queries. As demonstrated in Figure 1b, GPT-4 generates syntactically valid but semantically incorrect queries that fail to be pruned by existing rule-based syntax correction strategies.

Against these challenges, this paper formally defines the Text-to-MMQL task and proposes an adaptive and robust Text-to-MMQL framework. To overcome the dataset scarcity problem, we design a specialized Text-to-MMQL dataset with complex multi-model data schemas and typical MMQLs. Our framework incorporates a holistic representation for multi-model schemas, an adaptive intermediate representation for diverse MMQLs with error-correction strategies for robust generation.

The main contributions of this paper are summarized as follows.

1) This paper establishes a formal task definition for Text-to-MMQL (Section 2), which has not been previously presented in the literature.

2) To the best of our knowledge, this paper develops the first Text-to-MMQL dataset that incorporates three diverse MMQLs (Section 3).

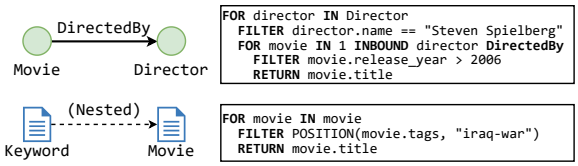
3) This paper proposes MMTrans, an adaptive and robust Text-to-MMQL framework (Section 4). MMTrans leverages a multi-model schema embedding module and an MMQL intermediate representation strategy to handle heterogeneous schemas and diverse MMQL grammars.

4) Comprehensive experimental results on the proposed dataset show the state-of-the-art performance of MMTrans, which improves end-to-end Text-to-MMQL accuracy by more than 9% compared with our adapted baseline methods (Section 5).

2 Preliminaries

In this section, we give definitions related to MMQLs and the Text-to-MMQL problem. For a target MMQL $\mathcal{P}$, we denote the corresponding database as $D_{\mathcal{P}}$, which accepts queries written in the grammar of $\mathcal{P}$ and outputs the execution results. The multi-model schema information $\mathcal{S}$ consists of different schemas from diverse data models. Therefore, we represent $\mathcal{S}$ as a list of pairs: $\mathcal{S} = [(\sigma_1, S_1), \cdots, (\sigma_{|\mathcal{S}|}, S_{|\mathcal{S}|})]$, where σ_i is the i-th scope (name of data model) and S_i is the schema representation within σ_i. Definition 1 formally defines the Text-to-MMQL problem.

Definition 1 (Text-to-MMQL). *Given a natural language question $Q = (q_1, \cdots, q_{|Q|})$ as a sequence of tokens, the target MMQL $\mathcal{P}$, and the corresponding multi-model data schema $\mathcal{S}$, the goal of Text-to-MMQL problem is to generate a query $y_{\mathcal{P}}$*




```
FOR director IN Director
  FILTER director.name == "Steven Spielberg"
  FOR movie IN 1 INBOUND director DirectedBy
    FILTER movie.release_year > 2006
    RETURN movie.title
```

```
FOR movie IN movie
  FILTER POSITION(movie.tags, "iraq-war")
  RETURN movie.title
```

Figure 2: An example of relationships and nested arrays in the multi-model schema of IMDB dataset as well as AQL queries that access these schema items.

that conforms to the grammar of $\mathcal{P}$ and captures the semantic of Q.

Figure 1c illustrates a multi-model schema representation $\mathcal{S}$ corresponding to the schema shown in Figure 1a. For instance, the document schema is represented as (σ_3, S_3), where $\sigma_3 =$ `Document` and S_3 is a JSON object that describes the nested structure of the `Order` and the `Product` documents.

3 Text-to-MMQL Dataset

To evaluate the effectiveness of Text-to-MMQL approaches, we present, to the best of our knowledge, the first Text-to-MMQL dataset. Specifically, this dataset is adapted from the IMDB (Yaghmazadeh et al., 2017) dataset, originally designed for the Text-to-SQL task. IMDB is suitable for the multi-model scenario due to its complex multiple-table join queries, which can be effectively transformed into multi-model queries in MMQL.

The construction of the Text-to-MMQL dataset involves two steps: (1) we convert the original relational schema into a multi-model schema (see Subsection 3.1), and (2) we prepare MMQL queries corresponding to the natural language questions (see Subsection 3.2).

3.1 Schema Conversion

We transform the IMDB dataset's relational schema into a multi-model schema, which is essential to evaluate Text-to-MMQL methods. The target multi-model schema incorporates *graph* and *document* data models besides the relational data model, as illustrated in Figure 2.

Graph. To convert the relational schema into the graph schema, we use vertices and edges to represent entity tables and edge tables, respectively. We first create vertex classes and edge classes in the graph schema, followed by creating graph vertices and edges that hold identical attributes compared to the relational tables. For instance, we transform each row in the `movie` table into a vertex with label `Movie`, and each row in the `Director` table into a vertex with label `Director`. We further transform each row in the `directed_by` edge table, which links movies to directors, into an edge with label `DirectedBy` that connects a `Movie` vertex to a `Director` vertex. The conversion procedure is further illustrated in Appendix A.1.

Document. We transform a part of tables into nested arrays or documents. We first create document collections in the document schema, followed by creating nested documents corresponding to the relational edge tables. For instance, each row of the `tags` edge table specifies a keyword of a movie. We transform all keywords associated with a movie into an array nested in the movie document. The conversion procedure is further illustrated in Appendix A.2.

Table 1: Statistics of the variants of the proposed Text-to-MMQL dataset.

Statistics	IMDB-AQL	IMDB-ECQL	IMDB-SQL++
Single-model Samples	33%	33%	33%
Multi-model Samples	67%	67%	67%
Average Tokens	52.17	75.54	38.18
Average Entities	2.78	3.28	2.66
Average Cross-model Joins in Multi-model Samples	1.33	2.02	1.03
Aggregates	25.80%	15.32%	4.84%
Sorts	8.87%	8.87%	8.87%
Nested Sub-queries	0.0%	67.7%	68.5%

3.2 MMQL Query Preparation

We implement multi-model queries in the IMDB dataset in three typical MMQLs, namely AQL, SQL++, and ECQL, whose characteristics are detailed in Section 6. Specifically, the gold SQL queries in the IMDB dataset are manually rewritten to conform to the grammar rules of the three MMQLs. This is based on the grammar diversity and the optimal expression considerations. First, different MMQLs have varied syntax structures, making direct query interpretation technically challenging. Second, the most appropriate implementation of a query often differs across MMQLs. Our manual rewriting ensures that the MMQLs leverage the structures that fit their corresponding styles.

Our dataset enables a comprehensive evaluation across unified grammar and composite grammar. While AQL and SQL++ adhere to unified grammar rules, ECQL combines grammar rules from various single-model query languages. To account for these differences, we structure them as distinct variants, denoted as IMDB-AQL, IMDB-SQL++, and

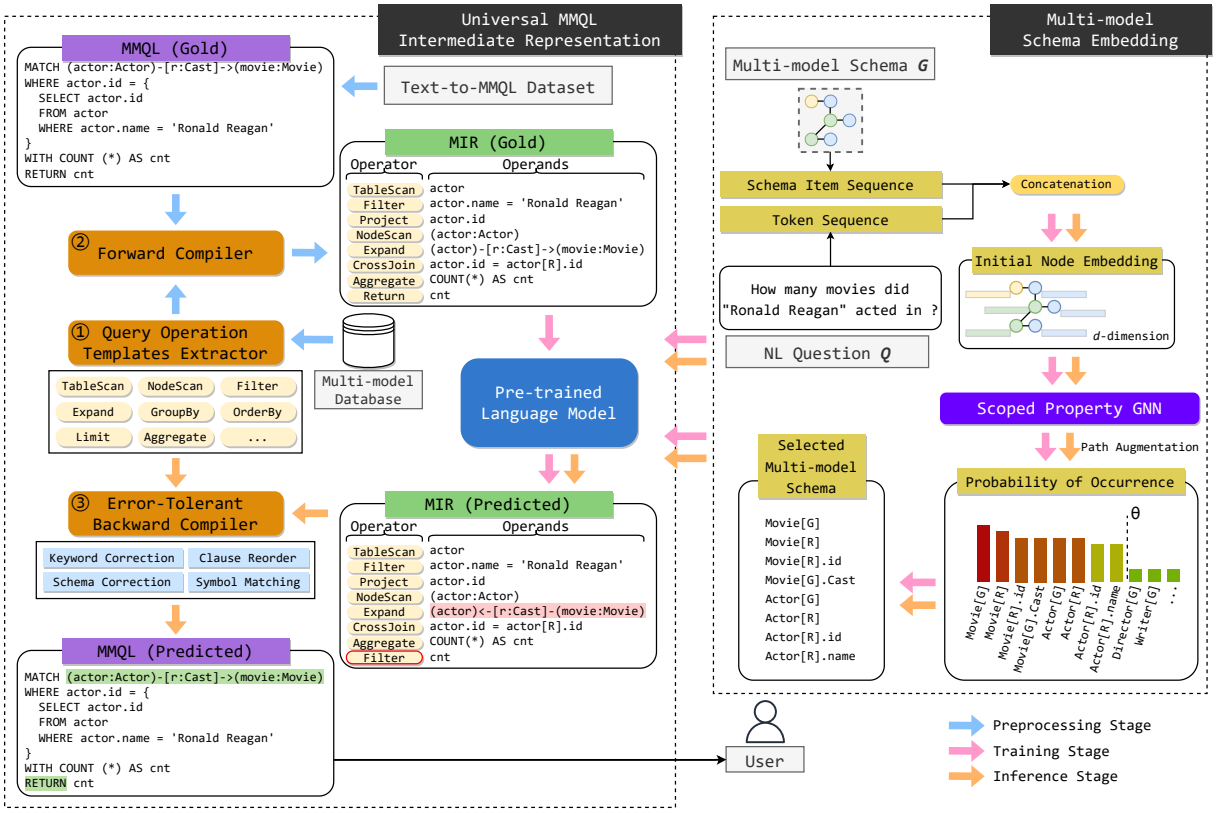


Figure 3: An overview of the proposed MMTrans framework. MIR is short for MMQL Intermediate Representation.

IMDB-ECQL respectively. Table 1 summarizes the statistics of the constructed Text-to-MMQL dataset. All variants share identical distributions of single-model (33%) and mutli-model (67%) samples, ensuring a detailed breakdown analysis over single-model queries or cross-model queries. Furthermore, all variants include more than one average cross-model joins, reflecting practical challenges in multi-model querying.

4 MMTrans Framework

As depicted in Figure 3, the proposed MMTrans framework addresses the *Schema Heterogeneity* problem through a Multi-model Schema Embedding method to effectively capture the multi-model schema information, and proposes a novel intermediate representation strategy for MMQLs.

The workflow of MMTrans consists of three stages. In the *preprocessing* stage, we automatically convert the gold MMQL queries into their intermediate representation as training and testing samples. In the *training* stage, we utilize the prepared samples to finetune the Pre-trained Language Model (PLM) based on sequence-to-sequence architecture, while training the multi-model schema embedding module. MMQL queries in training samples are represented with the proposed intermediate representation format. In the *inference* stage, MMTrans predicts the schema items related to the input natural language question Q and utilizes the PLM to predict a sequence in the intermediate representation format. The generated sequence is finally compiled to the MMQL format.

4.1 Multi-model Schema Embedding

We propose the multi-model schema embedding module against the *Schema Heterogeneity* challenge that the schema of different data models may adopt diverse representations. Specifically, we model the multi-model schema as a *Scoped Property Graph* (SPG) and propose a method based on Graph Neural Networks (GNNs) to effectively learn the multi-model schema embedding. We first give the definition of SPG in Definition 2, followed by the details of the proposed method.

Definition 2 (Scoped Property Graph). *A scoped property graph is denoted as* $G = (V, E, \Sigma)$*, where* V *and* E *are the sets of vertices and edges, respectively, and* $\Sigma = \{\sigma_1, \cdots, \sigma_{|\mathcal{S}|}\}$ *represents the set of scopes. Each vertex* $v \in V$ *is a tuple* $v = (l, \sigma)$*, where* l *is the label of* v*,* $\sigma \in \Sigma$ *is the scope of* v*. Each directed edge* $e \in E$ *is a tuple* $e = (v_s, v_t)$*,*

where $v_s, v_t \in V$ are the source and target vertices, respectively.

SPG offers a significant advantage in modeling complex relationships within multi-model schemas. For instance, SPG can intuitively represent the relationships `Movie-DirectedBy-Director` with schema vertices, as depicted in Figure 2. In contrast, this schema relationship is difficult to capture with sequential schema representation methods, since the MMQLs under consideration do not explicitly declare foreign keys.

We model the multi-model schema items as the nodes of a global SPG and propose a Scope Property Graph Neural Network (SPGNN) framework that fuses the node features from the embedding features learned by the PLM. We also incorporate a path augmentation strategy to encourage the PLM to predict schema items that form a connected subgraph. In the training stage, we first leverage the PLM to obtain the embeddings of all schema items in the context of question Q. These embeddings are used as the initial node embeddings of the SPGNN, which is trained to predict the probability of occurrence of each schema item. In the inference stage, we select the schema items whose probability of occurrence is greater than the threshold θ_s.

Initial node embedding. Given the natural language question Q and the SPG G, we construct a token sequence composed of the tokens in Q and the labels of the vertices in G. The embedding $Q_{emb} \in \mathbb{R}^{(|Q|+|V|)\times d}$ is calculated as:

$$Q_e = \mathcal{E}\left(Q \| v_1.\sigma \| v_1.l \| \cdots \| v_{|V|}.\sigma \| v_{|V|}.l\right), \quad (1)$$

where $\mathcal{E}(\cdot)$ denotes the embedding function for a token sequence, d denotes the embedding dimension, and $\|$ denotes vector concatenation. Various pre-trained embedding models can be adopted.

Schema embedding GNN. The GNN model is trained to perform a binary classification for each node v, where the two categories indicate *occurrence* and *absence* respectively. The Negative Log Likelihood (NLL) loss is adopted to measure the discrepancy between the predicted and the real occurrences of schema items.

Path augmentation. Based on the observation that the nodes which actually appear in queries are expected to form a connected subgraph of G, we propose the path augmentation strategy to encourage the GNN model to classify connected nodes as occurring. Specifically, for one node v_i, the probability of occurrence of every neighbor node v_j is aggregated to v_i, as long as either of the following conditions holds: (1) Belonging condition: v_i represents an entity in its data model and v_j represents an attribute of this entity, and (2) Join condition: Both v_i and v_j represent entities and there exists another neighbor of v_i, as denoted by v_k, that satisfies v_k represents an entity and $v_k \neq v_j$. Given $\mathcal{P} = [p_1^+, \cdots, p_{|V|}^+, p_1^-, \cdots, p_{|V|}^-]$ where p_i^+ and p_i^- are the probability of occurrence and absence respectively, the aggregated probability of occurrence of v_i is calculated as:

$$\mathrm{Agg}(p_i^+) = p_i^+ + \sum_{\substack{v_j \in \mathrm{Neighbor}(v_i) \wedge \\ (B(v_i,v_j) \vee J(v_i,v_j))}} p_j^+ \quad (2)$$

$$p_i^{+\prime}, p_i^{-\prime} = \mathrm{Softmax}(\mathrm{Agg}(p_i^+), p_i^-), \quad (3)$$

where $B(\cdot)$ and $J(\cdot)$ represent the belonging condition and the join condition, respectively.

4.2 Adaptive MMQL Intermediate Representation

To address the *Language Heterogeneity* challenge, this paper proposes MIR, an adaptive MMQL Intermediate Representation strategy. MIR enhances Text-to-MMQL performance with a succinct representation format, automatic query operation template extraction, and robust rule-based query correction.

Overview. As illustrated in Figure 3, the proposed MIR consists of three stages. During the preprocessing stage, we automatically extract the query operation templates corresponding to the target multi-model database. The forward compiler compiles each gold query in the MMQL dataset into its MIR format. The preprocessed dataset is then utilized to finetune the PLM in the training stage. In the inference stage, for each query in MIR format predicted by the PLM, the error-tolerant backward compiler performs proposed error correction rules and compiles the query back to its MMQL format.

The core idea of the proposed strategy is to utilize the serialized logical query plans in PLM training and inference. Benefiting from this design, MMTrans transforms MMQL queries into succinct form and therefore reduces token numbers. Besides, MIR format provides an opportunity to perform robust error correction on the generated queries for various MMQLs, not limited by the specific grammars.

4.2.1 Forward Compilation

In the preprocessing stage, we compile each gold MMQL query $y_{\mathcal{P}}$ to its MIR format $r_{\mathcal{P}}$ and extract its skeleton $s_{\mathcal{P}}$. Specifically, we first extract the query operator templates according to the query parser of $D_{\mathcal{P}}$. Each query operator template t consists of an operator name op and an operand number n. We then specify the representation of the logical query plan corresponding to $D_{\mathcal{P}}$ as Definition 3.

Definition 3 (Logical Query Plan)**.** *A logical query plan is a binary tree whose nodes are query operators. Each query operator o is a 3-tuple $o = (op, t, m)$, where op is the operator name, t is the template corresponding to o, and m is the number of children of o.*

For each logical query plan, we obtain its postorder traversal sequence as $r_{\mathcal{P}}$, with $s_{\mathcal{P}}$ by concatenating $o.op$ for each node in $r_{\mathcal{P}}$. The compilation algorithm is analyzed in detail in Appendix B.1.

4.2.2 Error-Tolerant Backward Compilation

To enhance the robustness of MMTrans, we propose a rule-based error correction strategy that reduces the error-predicted tokens by PLM. We present four categories of the proposed correction rules, namely Keyword Correction, Schema Correction, Clause Reorder, and Symbol Matching. Furthermore, we propose an execution-level lossless algorithm that compiles the MIR query backward to its MMQL format.

Keyword Correction (KC). KC aims to detect wrong predicted keywords and correct them. Specifically, for each literal value v in the predicted MIR query and each keyword w in question Q, if the word-level similarity between v and w is greater than the threshold θ_{q}, then v is replaced by w.

Schema Correction (SC). SC aims to correct the predicted clauses that violate the multi-model schema observed from $D_{\mathcal{P}}$.

Clause Reorder (CR). CR aims to recognize clauses that are invalid in terms of multi-model database execution logic. CR rules correct a MIR by removing or reordering invalid clauses. For example, a `Return` clause in an AQL query is always expected to be the final clause.

Symbol Matching (SM). SM aims to identify and correct the clauses with error-predicted operator names or operands. SM also recognizes the clauses whose operands do not conform to any operator template, seeking for the most similar operator template and filling the operands with default values.

Backward Compilation (BC) algorithm. After performing correction rules on the predicted MIR query $\hat{r}_{\mathcal{P}}$, we finally compile $\hat{r}_{\mathcal{P}}$ backward to MMQL query $\hat{y}_{\mathcal{P}}$. The BC algorithm reconstructs a logical query plan from its postorder traversal sequence. Benefiting from our query operator template extraction, we can prove that the proposed algorithm is *plan-level lossless*: if the MIR query is accurate, i.e. $\hat{r}_{\mathcal{P}} = r_{\mathcal{P}}$, then the BC algorithm guarantees to reconstruct $\hat{y}_{\mathcal{P}}$ with the same logical query plan as that of $y_{\mathcal{P}}$. The BC algorithm is analyzed in detail in Appendix B.2.

5 Experimental Results

5.1 Experimental Setup

Environments. All experiments are conducted on a Linux server with 10-core Intel Xeon E5 CPU with 320GB memory and one NVIDIA GeForce RTX 3090 GPU with 24GB memory. The operating system is Ubuntu 20.04.5 LTS version.

Datasets. We conduct experiments on the dataset variants proposed in Section 3. The dataset variants are named *IMDB-AQL*, *IMDB-SQL++*, and *IMDB-ECQL* to avoid ambiguity. We adopt k-fold cross-validation on these datasets.

Multi-model databases and MMQLs. We set up multi-model database environments to evaluate MMQL queries. For AQL (ara, 2025) queries, we adopt ArangoDB v3.10.5-community. For ECQL (Shi et al., 2024a) queries, we implement a query processor on top of PostgreSQL (pos, 2025) v13.4 and Neo4j (neo, 2025) v3.5.22 as database engines. Relational data and graph data are stored in PostgreSQL and Neo4j, respectively. Besides, document data are stored as PostgreSQL's JSONB format. Multi-model data are imported to the multi-model databases in advance. We implement the corresponding drivers in Python v3.9.20 style. In determining the EX matching of two MMQL queries, we compare their corresponding execution results in Python data types. Our supplementary materials can be accessed at `https://github.com/stone-ts15/text_to_mmql`.

Baseline approaches. We mainly consider three categories of baseline approaches. First, we adapt Seq2Seq (Dong and Lapata, 2016) to the Text-to-MMQL task. For PLMs, we select BART (Lewis, 2019) and T5 (Raffel et al., 2020) which both adopt the transformer architecture. We also compare LLM-based approaches. For zero-shot methods, we adopt GPT-4/GPT-4o (Achiam et al., 2023). We

Table 2: Overall Text-to-MMQL translation accuracy on the proposed datasets.

	Method	IMDB-AQL		IMDB-ECQL		IMDB-SQL++	
		LF (%)	EX (%)	LF (%)	EX (%)	LF (%)	EX (%)
Seq2Seq Based	Seq2Seq	0.81	0.81	0.74	0.74	2.35	2.35
	Seq2Seq + Attn.	3.15	4.70	3.96	6.31	3.08	3.08
PLM Based	BART-large	8.52	11.74	1.54	1.54	6.10	6.90
	T5-base	69.33	73.29	58.40	64.85	22.82	23.62
	T5-large	73.36	75.64	57.52	65.51	22.01	24.36
LLM Based	GPT-4o (Zero Shot)	10.20	40.80	0.0	19.59	0.0	3.96
	GPT-4 (Zero Shot)	5.05	43.07	0.0	20.47	0.0	4.77
	PURPLE-GPT4o	56.78	62.29	52.82	62.43	34.63	49.60
	PURPLE-GPT4	56.85	59.20	52.02	59.20	34.56	51.21
Proposed	MMTrans-base	79.67	**85.10**	**67.86**	**77.32**	**69.54**	**80.48**
	MMTrans-large	**82.83**	**85.10**	67.72	74.83	64.71	74.91

also implement PURPLE (Ren et al., 2024) which selects queries similar to the input question from the training dataset as demonstrations. It should be noted that most of the state-of-the-art approaches for Text-to-SQL task are excluded from our experiments due to their SQL-specific optimization strategies, which are not directly applicable to MMQLs.

Metrics. We adopt two typical evaluation metrics: (1) Logical Form (LF) accuracy that compares the predicted query with its corresponding gold query exactly at word level, and (2) Execution (EX) accuracy that compares the execution result of the predicted query with its corresponding gold query.

5.2 Overall Performance Evaluation Results

Table 2 reports overall evaluation results on the proposed datasets. MMTrans outperforms the compared approaches by more than 10.42% in LF and 9.46% in EX on IMDB-AQL, and 9.46% in LF and 12.47% in EX on IMDB-ECQL, showing the effectiveness of MMTrans. Most of the compared approaches present lower accuracy on IMDB-ECQL compared to IMDB-AQL, indicating the complexity of grammar of ECQL. Although LLM-based methods achieve a slightly higher accuracy on single-model query prediction, they show lower overall performance on both datasets. This is mainly because LLMs are not trained with adequate materials in terms of the target MMQLs and only learn limited grammar information. Although PURPLE introduces a demonstration approach in prompt construction, it still fails to learn accurate patterns from these demonstrations.

We also observe that larger models do not result in an performance improvement. For instance, on IMDB-ECQL, the LF accuracy of T5-large (57.52%) is 0.88% lower than that of T5-base (58.40%), showing that the difficulties introduced in Text-to-MMQL problem can hardly be eliminated by larger models. Without loss of generality, we select T5-base as the PLM in the following experiments, which mainly aim to investigate the effectiveness of our proposed components.

Table 3: Ablation studies on multi-model schema embedding and MIR.

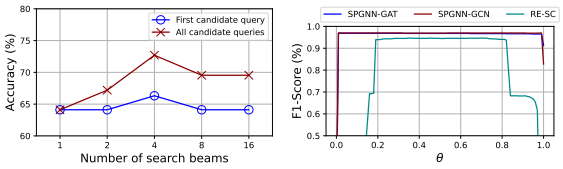
Method	AQL		ECQL		SQL++	
	LF	EX	LF	EX	LF	EX
MMTrans	**79.67**	**85.10**	**67.86**	**77.32**	69.54	**80.48**
Results for multi-model schema embedding						
Full Schema	70.42	75.05	56.78	64.63	54.58	64.78
No Schema	69.54	75.71	56.71	63.83	50.62	57.73
Results for MMQL intermediate representation						
-w/o. KC	76.52	81.95	66.25	75.71	66.39	77.32
-w/o. SC	77.32	82.76	64.63	76.52	67.20	78.87
-w/o. CR	77.32	82.76	67.13	76.59	68.74	79.67
-w/o. SM	76.73	81.43	67.86	77.32	66.46	77.40
-w/o. ETBC	70.35	75.05	62.29	74.17	59.27	73.36
-w/o. Skeleton	77.47	80.55	67.79	74.98	**70.21**	78.80
-w/o. ETBC /Skeleton	74.31	77.40	61.41	70.21	65.51	76.52
-w/o. MIR	71.75	74.83	59.94	66.98	21.13	25.17

5.3 Ablation Studies

Multi-model schema embedding. Table 3 (upper) reports the evaluation results for the multi-model schema embedding. For a fair comparison, all compared approaches are trained on the MIR format without skeletons and error-correction strategies. The evaluation results show that the multi-model schema embedding module brings performance improvement by 4.77% LF and 1.69% EX

Table 4: Comparison of schema item prediction accuracy.

	Method	AUC	Max F1-Score	Parameter Number
Fix PLM	RE-SC	0.9324	0.6080	18M
	SPGNN-GAT	0.9694	0.8060	49K
	SPGNN-GCN	0.9809	0.8308	49K
Finetune PLM	RE-SC	0.9457	0.9465	18M
	SPGNN-GAT	0.9935	0.9691	49K
	SPGNN-GCN	**0.9939**	**0.9705**	49K



(a) Varying number of beams in translation. (b) Varying θ_s in schema item prediction.

Figure 4: Results on the impact of parameters.

on IMDB-AQL, as well as 4.70% LF and 6.38% EX on IMDB-ECQL. We also observe that simply using full schema sequence would not greatly improve the prediction performance and even decrease EX accuracy from 75.71% to 75.05% on IMDB-AQL. One possible reason is that the full schema sequence increases the input token length without providing the schema information that is closely relevant to the input question.

We also report the schema item prediction accuracy in F1-Score and area under curve of receiver operating characteristic curve (ROC-AUC) metrics in Table 4. We compare Graph Convolutional Network (GCN) and Graph Attention Network (GAT) as the GNN implementation inside our proposed SPGNN. We also implement the schema classifier of RESDSQL for schema item prediction, as denoted by RE-SC. The results show that GCN and GAT achieve comparable highest AUC score and F1-score, outperforming RE-SC under both fix-PLM and finetune-PLM situations. Furthermore, the model size in terms of parameter number of SPGNN is significantly (more than two magnitudes) smaller than that of RE-SC, enabling efficient training for schema item prediction.

MIR. Table 3 (lower) reports the evaluation results for MIR. All approaches compared are equipped with multi-model schema embedding. The results show that by introducing MIR, MMTrans achieves more than 7% overall accuracy improvements, indicating that introducing a succinct intermediate representation can effectively reduce the impact of the complexity of MMQL grammar rules.

Table 5: Study on the scalability of MIR.

Metric	Method	AQL	ECQL	SQL++
Average Tokens	Raw	52.17	75.54	38.18
	MIR	**43.46**	**68.69**	**25.95**
End-to-end Prediction Time (ms)	Raw	107.18	101.22	109.43
	MIR	**102.35**	**90.47**	**107.17**
	- Inference	102.19	90.32	106.87
	- ETBC	0.16	0.15	0.30

5.4 Parameter Sensitivity and Scalability

Parameter sensitivity. Figure 4a reports the prediction LF accuracy under various searching beams of the candidates generated by PLM. The results show that MMTrans achieves the highest accuracy at 4 beams, while increasing the number of searching beams does not improve accuracy. Consequently, we adopt 4 searching beams in our major experiment settings. Figure 4b reports the F1-scores of schema item prediction under various threshold θ_s, showing that SPGNN achieves robust schema item prediction where the value of θ_s does not significantly influence the accuracy. Specifically, SPGNN-GAT and SPGNN-GCN ensure F1-scores no less than 0.9648 and 0.9689, respectively.

Scalability. We measure the token lengths and inference time of MIR. As shown in Table 5, queries written in MIR format are 16.70% shorter than the raw format in terms of token length, which results in 4.51% end-to-end inference time speedup on IMDB-AQL. For IMDB-ECQL, MIR queries are 9.07% shorter, which results in 10.62% end-to-end inference time speedup. The results show that the time cost of ETBC is dominated by the PLM inference time.

6 Related Work

In this section, we briefly review the research efforts on natural language to query language translation. Besides, we summarize studies on multi-model schema representations and MMQLs.

Natural Language to Query Language Translation. Significant developments are made in the Text-to-SQL task with the studies of deep learning technologies. Modern PLM-based Text-to-SQL methods (Wang et al., 2020; Scholak et al., 2021; Hui et al., 2022; Li et al., 2023a,b) adopt a sequence-to-sequence architecture. Semantic

rules (Fu et al., 2023) and intermediate representations (Gan et al., 2021) are studied to improve prediction accuracy. However, most of the learning-based Text-to-SQL approaches strongly rely on the SQL grammar and are not applicable for MMQLs. LLMs show promise in Text-to-SQL tasks in recent studies ((Li et al., 2024a)), including zero-shot methods (Dong et al., 2023; Liu et al., 2023) and few-shot approaches (Pourreza and Rafiei, 2023, 2024; Ren et al., 2024) which search query samples most related to the input question as demonstrations. However, LLM-based approaches suffer from a performance degradation in Text-to-MMQL task due to inadequate data related to target MMQLs. The adaptation for existing Text-to-SQL methods to the Text-to-MMQL task relies on Text-to-MMQL datasets with high quality. Existing Text-to-SQL datasets, such as Spider (Yu et al., 2018), BIRD (Li et al., 2024b), and Spider 2.0 (Lei et al., 2024), involve complex SQL constructs, including geometric functions, branch statements, and other advanced features. Consequently, it is challenging to interpret these SQL queries into equivalent MMQL representations.

There are also studies that focus on the translation from natural language to NoSQL query languages. SPBERT (Tran et al., 2021) aims to translate questions into SPARQL (Pérez et al., 2009) queries based on BERT (Devlin et al., 2018). The Seq2Seq (Dong and Lapata, 2016; Gu et al., 2016) model is evaluated on a Text-to-Cypher dataset SpCQL (Guo et al., 2022), showing the challenges for Text-to-Cypher task with complex grammars.

MMQLs and Multi-model Schema Representation. Existing multi-model databases and polystores usually design their query languages based on their own holistic data models. An abstract data model based on category theory is implemented as a multi-model query language (Čontoš, 2021; Holubova et al., 2021; Uotila et al., 2021). ArangoDB's AQL (ara, 2025) is designed based on the link-document structure for graph and document data. As adopted by many databases (ori, 2025; ast, 2025), SQL++ (Ong et al., 2014) query language is proposed based on the data model that includes JSON data apart from relational data, while gSQL++ (Glenn and Carey, 2024) further extends SQL++ to support graph operations. There are also approaches that incorporate multiple interfaces from different data models to provide a holistic MMQL. CloudMdsQL (Kolev et al., 2016) and ECQL (Shi et al., 2024a) combine sub-queries written in different single-model query languages. Polypheny-DB (Vogt et al., 2018, 2020) accepts single-model query languages corresponding to its data stores. The query language adopted in XDB (Gavriilidis et al., 2023a,b) extends SQL with key-value operations. There are existing studies on MMQL query generation for benchmarking purpose, including template-based generation methods (Cheng et al., 2022; Shi et al., 2024b) and LLM-based generation methods (Zheng et al., 2024). However, these approaches are not capable for Text-to-MMQL translation. To the best of our knowledge, there do not exist approaches for Text-to-MMQL translation.

7 Conclusion and Future Work

In this paper, we formally define the Text-to-MMQL task and propose the first Text-to-MMQL dataset. We propose an adaptive and robust Text-to-MMQL framework that includes a novel multi-model schema embedding module as well as an MMQL intermediate representation that enables succinct query formats and rule-based error correction. Experimental results show that MMTrans improves the Text-to-MMQL performance compared with the adapted baseline methods.

Our future work includes validation of Text-to-MMQL approaches on larger datasets. For the purpose of adapting multi-domain Text-to-SQL datasets such as Spider and Spider 2.0, an extensible schema conversion is in need to automatically and reasonably schedule the multi-model schema according to the original relational schema.

Limitation

Although our experimental evaluation shows the priority of the proposed framework, there are still limitations within this paper. As a start, our Text-to-MMQL dataset is proposed on top of a single-domain dataset with three MMQLs, while the construction of multi-domain Text-to-MMQL datasets remains a challenging problem that needs to be further considered. Besides, our proposed framework is only responsible for supervised Text-to-MMQL translation.

Acknowledgments

This work is supported in part by the National Natural Science Foundation of China (No. 62372264, 62021002, 92467203). Chaokun Wang is the corresponding author.

References

2025. Arangodb. [Online; accessed May 30, 2025].

2025. Asterixdb. [Online; accessed May 30, 2025].

2025. Neo4j. [Online; accessed May 30, 2025].

2025. Orientdb. [Online; accessed May 30, 2025].

2025. Postgresql. [Online; accessed May 30, 2025].

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

Audrey Cheng, Xiao Shi, Aaron Kabcenell, Shilpa Lawande, Hamza Qadeer, Jason Chan, Harrison Tin, Ryan Zhao, Peter Bailis, Mahesh Balakrishnan, et al. 2022. Taobench: An end-to-end benchmark for social network workloads. *Proceedings of the VLDB Endowment*, 15(9):1965–1977.

Pavel Čontoš. 2021. Abstract model for multi-model data. In *International Conference on Database Systems for Advanced Applications*, pages 647–651. Springer.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 33–43, Berlin, Germany. Association for Computational Linguistics.

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

Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. 2023. Catsql: Towards real world natural language to sql applications. *Proceedings of the VLDB Endowment*, 16(6):1534–1547.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R Woodward, John Drake, and Qiaofu Zhang. 2021. Natural sql: Making sql easier to infer from natural language specifications. In *Findings of the Association for Computational Linguistics: EMNLP 2021*, pages 2030–2042.

Haralampos Gavriilidis, Kaustubh Beedkar, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2023a. In-situ cross-database query processing. In *39th IEEE International Conference on Data Engineering, ICDE 2023, Anaheim, CA, USA, April 3-7, 2023*, pages 2794–2807. IEEE.

Haralampos Gavriilidis, Leonhard Rose, Joel Ziegler, Kaustubh Beedkar, Jorge-Arnulfo Quiané-Ruiz, and Volker Markl. 2023b. XDB in action: Decentralized cross-database query processing for black-box DBMSes. *Proc. VLDB Endow.*, 16(12):4078–4081.

Galvizo Glenn and Michael J. Carey. 2024. Graphix: "One user's JSON is another user's graph". In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. IEEE.

Jiatao Gu, Zhengdong Lu, Hang Li, and Victor O.K. Li. 2016. Incorporating copying mechanism in sequence-to-sequence learning. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1631–1640, Berlin, Germany. Association for Computational Linguistics.

Aibo Guo, Xinyi Li, Guanchen Xiao, Zhen Tan, and Xiang Zhao. 2022. Spcql: A semantic parsing dataset for converting natural language into cypher. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management, 2022*, pages 3973–3977. ACM.

Irena Holubova, Pavel Contos, and Martin Svoboda. 2021. Categorical management of multi-model data. In *25th International Database Engineering & Applications Symposium*, pages 134–140.

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Yanyang Li, Bowen Li, Jian Sun, and Yongbin Li. 2022. S2sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers. In *Findings of the Association for Computational Linguistics: ACL 2022*, pages 1254–1262.

Boyan Kolev, Carlyna Bondiombouy, Patrick Valduriez, Ricardo Jiménez-Peris, Raquel Pau, and José Pereira. 2016. The CloudMdsQL multistore system. In *Proceedings of the 2016 International Conference on Management of Data*, pages 2113–2116.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

M Lewis. 2019. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3):127.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13076–13084.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

Kian Win Ong, Yannis Papakonstantinou, and Romain Vernoux. 2014. The SQL++ unifying semi-structured query language, and an expressiveness benchmark of SQL-on-Hadoop, NoSQL and NewSQL databases. *CoRR, abs/1405.3631*.

Beng Chin Ooi, Gang Chen, Mike Zheng Shou, Kian-Lee Tan, Anthony Tung, Xiaokui Xiao, James Wei Luen Yip, Bingxue Zhang, and Meihui Zhang. 2023. The metaverse data deluge: What can we do about it? In *2023 IEEE 39th International Conference on Data Engineering (ICDE)*, pages 3675–3687. IEEE.

Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. 2009. Semantics and complexity of sparql. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Mohammadreza Pourreza and Davood Rafiei. 2024. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67.

Tonghui Ren, Yuankai Fan, Zhenying He, Ren Huang, Jiaqi Dai, Can Huang, Yinan Jing, Kai Zhang, Yifan Yang, and X. Sean Wang. 2024. PURPLE: making a large language model a better SQL writer. In *40th IEEE International Conference on Data Engineering, ICDE 2024, Utrecht, The Netherlands, May 13-16, 2024*, pages 15–28. IEEE.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 9895–9901.

Gengyuan Shi, Chaokun Wang, and Yabin Liu. 2024a. Ecql: Towards succinct and extensible modeling of multi-model query results. In *International Conference on Conceptual Modeling*, pages 112–130. Springer.

Gengyuan Shi, Chaokun Wang, Minghao Zhang, and Binbin Wang. 2024b. Font: a flexible polystore evaluation platform. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*, pages 5489–5492. IEEE.

Michael Stonebraker and Ugur Çetintemel. 2005. "one size fits all": An idea whose time has come and gone (abstract). In *Proceedings of the 21st International Conference on Data Engineering, ICDE 2005, 5-8 April 2005, Tokyo, Japan*, pages 2–11.

Hieu Tran, Long Phan, James T. Anibal, Binh T. Nguyen, and Truong-Son Nguyen. 2021. SPBERT: an efficient pre-training BERT on SPARQL queries for question answering over knowledge graphs. In *Neural Information Processing - 28th International Conference, ICONIP 2021, Proceedings, Part I*, volume 13108 of *Lecture Notes in Computer Science*, pages 512–523. Springer.

Valter Uotila, Jiaheng Lu, Dieter Gawlick, Zhen Hua Liu, Souripriya Das, and Gregory Pogossiants. 2021. Multi-model query processing meets category theory and functional programming. In *SEA-Data@ VLDB*, pages 48–49.

Marco Vogt, Nils Hansen, Jan Schönholz, David Lengweiler, Isabel Geissmann, Sebastian Philipp, Alexander Stiemer, and Heiko Schuldt. 2020. Polypheny-DB: Towards bridging the gap between polystores and HTAP systems. In *Heterogeneous Data Management, Polystores, and Analytics for Healthcare - VLDB Workshops, Poly 2020 and DMAH 2020, August 31 - September 4, 2020*, volume 12633 of *Lecture Notes in Computer Science*, pages 25–36. Springer.

Marco Vogt, Alexander Stiemer, and Heiko Schuldt. 2018. Polypheny-DB: Towards a distributed and self-adaptive polystore. In *2018 IEEE International Conference on Big Data (Big Data)*, pages 3364–3373. IEEE.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: query synthesis from natural language. *Proc. ACM Program. Lang.*, 1(OOPSLA):63:1–63:26.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 2018*, pages 3911–3921. Association for Computational Linguistics.

Chao Zhang and Jiaheng Lu. 2021. Holistic evaluation in multi-model databases benchmarking. *Distributed and Parallel Databases*, 39(1):1–33.

Chao Zhang, Jiaheng Lu, Pengfei Xu, and Yuxing Chen. 2018. UniBench: A benchmark for multi-model database management systems. In *Technology Conference on Performance Evaluation and Benchmarking*, pages 7–23. Springer.

Xiuwen Zheng, Arun Kumar, and Amarnath Gupta. 2024. Generating cross-model analytics workloads using llms. In *Proceedings of the 33rd ACM International Conference on Information and Knowledge Management*, pages 4303–4307.

A Schema Conversion

We transform the relational schema of the original IMDB dataset into a comprehensive multi-model schema. The target multi-model schema incorporates graph and document data models besides the relational data model.

A.1 Conversion to Graph Schema

The conversion from an entity table to graph vertices takes the following steps. We take the conversion from the `movie` table to the `Movie` vertex class as an example. First, we create a vertex class corresponding to the entity table (e.g., creating the `Movie` vertex class corresponding to the `movie` table). Second, for each tuple in the entity table, we create a vertex whose identification key corresponds to the primary key attribute value of this tuple (e.g. creating vertex `Movie {_id: 1}` corresponding to the tuple in the `movie` table where `msid = 1`). Third, for other columns in the tuple, we copy their attribute values to the created vertex.

The conversion from an edge table to graph edges takes the following steps. We take the conversion from the `cast` table to the `Cast` edge class as an example. First, create an edge class corresponding to the edge table (e.g. creating the `Cast` edge class corresponding to the `cast` table). Second, for each tuple in the edge table, find the source vertex and the target vertex whose identification keys are equal to the foreign keys in this tuple, respectively (e.g. finding source vertex `Actor {_id: 4}` and target vertex `Movie {_id: 2}`). Third, create an edge, which belongs to the edge class corresponding to this edge table, from the source vertex to the target vertex (e.g. creating edge `r: (Actor {_id: 4})-[r:Cast]->(Movie {_id: 2})`). Finally, for other columns in the tuple, copy their attribute values to the created edge.

Algorithm 1: Forward Compilation

Input: Target multi-model database P, query $y_{\mathcal{P}}$, and $delimiter$.
Output: Intermediate representation $r_{\mathcal{P}}$ and skeleton $s_{\mathcal{P}}$.

$plan \leftarrow P$.parse(q) ;
$C \leftarrow$ `PostorderTraversal(`*plan*`)` ;
$r_{\mathcal{P}} \leftarrow$ empty list ;
$s_{\mathcal{P}} \leftarrow$ empty list ;
for c **in** C **do**
  $t \leftarrow$ template corresponding to the operator $c.op$;
  $s_{\mathcal{P}} \leftarrow s_{\mathcal{P}} + [c.op, \text{“\_”}, delimiter]$;
  $r_{\mathcal{P}} \leftarrow r_{\mathcal{P}} + [c.op]$;
  for $i = 1$ **to** $t.n$ **do**
    $r_{\mathcal{P}} \leftarrow r_{\mathcal{P}} + [c.operands[i]]$;
  end
  $r_{\mathcal{P}} \leftarrow r_{\mathcal{P}} + [k]$;
end
return $r_{\mathcal{P}}, s_{\mathcal{P}}$;

A.2 Conversion to Document Schema

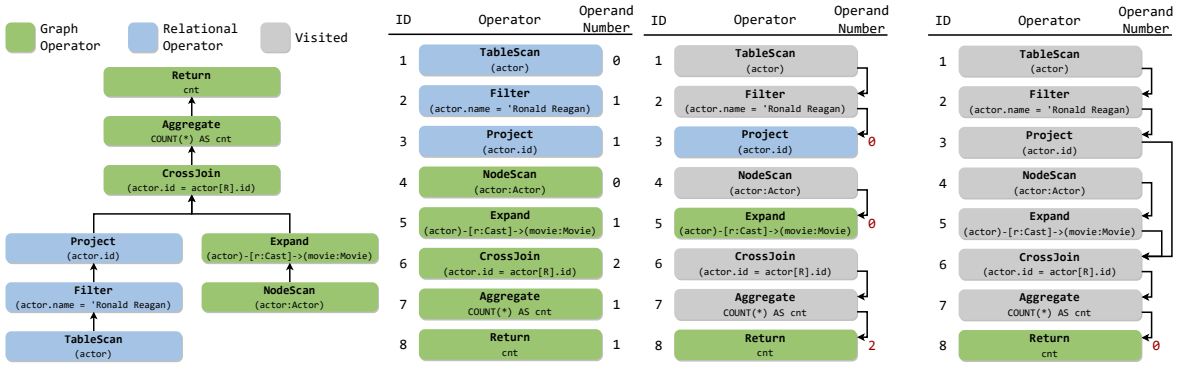
The conversion from an entity table and an edge table to arrays nested in documents takes the following steps. We take the conversion from the `keyword` table and the `tags` table to arrays nested in the `Movie` documents as an example. First, for each document in the document class, we add an attribute whose key equals the edge table name and value is an empty array (e.g. setting `movie.tags = []`). Second, for each tuple in the edge table, we find the document whose identification key is equal to the foreign key in this tuple (e.g. finding `movie` where `movie.id = 1`). Third, we find the tuple in the entity table whose primary key is equal to the foreign key in this edge tuple, convert this entity tuple into a document, and append this document in the array created in the first step (e.g. appending `Keyword {keyword: "iraq-war"}` to `movie.tags`).

B Algorithms Analysis

This section discusses the proposed forward compilation algorithm and backward compilation algorithm in detail, along with their time complexity analysis.

B.1 Forward Compilation

As shown in Algorithm 1, the forward compilation procedure first invokes the parser corresponding to MMQL P to obtain the logical query plan as a



(a) Logical query plan. (b) MIR query. (c) Backward compilation.

Figure 5: An example of MMQL intermediate representation of the query specified in Figure 3.

binary tree denoted as $plan$ (Line 1). The operator sequence C is then calculated as the post-order traversal of $plan$ (Line 2). $r_{\mathcal{P}}$ and $s_{\mathcal{P}}$ are initialized as empty lists (Lines 3-4). For each operator $c \in C$, its corresponding template t is identified through $c.op$ (Lines 5-6). The skeleton sequence of c is $[c.op, \text{“\_”}, delimiter]$ (Line 7). The representation of c consists of the operator name $c.op$, operands of number $t.m$, and the $delimiter$ (Lines 8-12). It is noteworthy that any symbol that does not bring syntactic ambiguity can be used as the delimiter. In this paper, we select the semicolon symbol (;), since it is not included in the original grammar rules of the MMQLs in our datasets.

Figure 5a shows the logical query plan corresponding to the gold MMQL query in Figure 3. It is noteworthy that although a query operator o may have various templates, its children number $o.m$ is uniquely determined by $o.op$. An example of compiled MIR is shown in Figure 5b, involving 8 different query operators. The extracted skeleton is “`TableScan _ ; Filter _ ; Project _ ; NodeScan _ ; Expand _ ; CrossJoin _ ; Aggregate _ ; Return _ ;`”. Compared with the raw MMQL query that involves a complex nested sub-query, the MIR query adopts a sequential structure with a list of clauses.

Complexity analysis. The time complexity of the forward compilation is $O(N \max(m))$, where N is the number of operators and $\max(m)$ is the largest number of operands.

B.2 Error-Tolerant Backward Compilation

The error-tolerant backward compilation procedure consists of error correction and backward compilation. First, correction rules are applied to correct potential errors in the MIR query $\hat{r}_{\mathcal{P}}$ predicted by the PLM. Second, the updated MIR query $\hat{r}_{\mathcal{P}}$ is compiled to the predicted MMQL query $\hat{y}_{\mathcal{P}}$.

The backward compilation algorithm guarantees that if the generated MIR query $\hat{r}_{\mathcal{P}}$ is equal to the gold MIR query $r_{\mathcal{P}}$, then the algorithm reconstructs $\hat{y}_{\mathcal{P}}$ with the same logical query plan as that of $y_{\mathcal{P}}$. Since the proposed MIR is a post-order traversal sequence of the original logical query plan, our goal is to prove that any non-empty binary tree T can be uniquely determined by its post-order traversal sequence, denoted by PT, given the number of children for each node. In particular, a node PT_i in PT is a *leaf node* if its corresponding operator takes no input operands, such as `Scan` operators. Considering that the number of children for a node of T can be 0, 1, or 2, we define two types of non-empty binary trees in the following proof procedure:

1) A $(0, 1, 2)$-tree, where each node has 0, 1, or 2 children nodes.

2) A $(0, 2)$-tree, where each node has 0 or 2 children nodes.

We first present Lemma 1 as follows.

Lemma 1. *Given PT as a post-order traversal sequence of a non-empty $(0, 2)$-tree T, for any longest sub-sequence of PT consisting of l consecutive leaf nodes starting from PT_i, denoted as $[PT_i, \cdots, PT_{i+l-1}]$, the nodes PT_{i+l-2} and PT_{i+l-1} must be sibling nodes in T.*

Proof. Assume, for contradiction, that PT_{i+l-2} and PT_{i+l-1} are not sibling nodes. (1) If PT_{i+l-1} is the right child of its parent, denoted as PT_{parent}, then the left child of PT_{parent} must exist (since T is a $(0, 2)$-tree) and must be exactly PT_{i+l-2},

according to the definition of post-order traversal. This outcome contradicts the assumption that PT_{i+l-2} and PT_{i+l-1} are not sibling nodes. (2) If PT_{i+l-1} is the left child of its parent node, denoted as PT_{parent}, then the right child of PT_{parent} must exist (since $\mathcal{T}$ is a $(0, 2)$-tree) and must be exactly PT_{i+l}, according to the definition of post-order traversal. However, this outcome would result in a longer sub-sequence of consecutive leaf nodes with length $l + 1$, which conflicts with the assumption that the longest sub-sequence of PT contains only l nodes. □

Based on Lemma 1, we present Theorem 1 as follows.

Theorem 1. *A non-empty $(0, 2)$-tree T can be uniquely determined by its post-order traversal sequence PT given the number of children for each node of T.*

Proof. Consider any longest sub-sequence of PT with l consecutive leaf nodes, namely $[PT_i, \cdots, PT_{i+l-1}]$. Given PT_{i+l-2} and PT_{i+l-1} are sibling nodes according to Lemma 1, the node PT_{i+l} is uniquely identified as the parent of PT_{i+l-2} and PT_{i+l-1}. By removing PT_{i+l-2} and PT_{i+l-1} from PT, a new sequence PT' of length $N - 2$ is obtained. This procedure can be repeated $\lfloor N/2 \rfloor$ times, progressively determining all parent-child relationships. □

We finally prove Theorem 2, which is equivalent to our main objective.

Theorem 2. *A non-empty $(0, 1, 2)$-tree T can be uniquely determined by its post-order traversal sequence PT given the number of children for each node of T.*

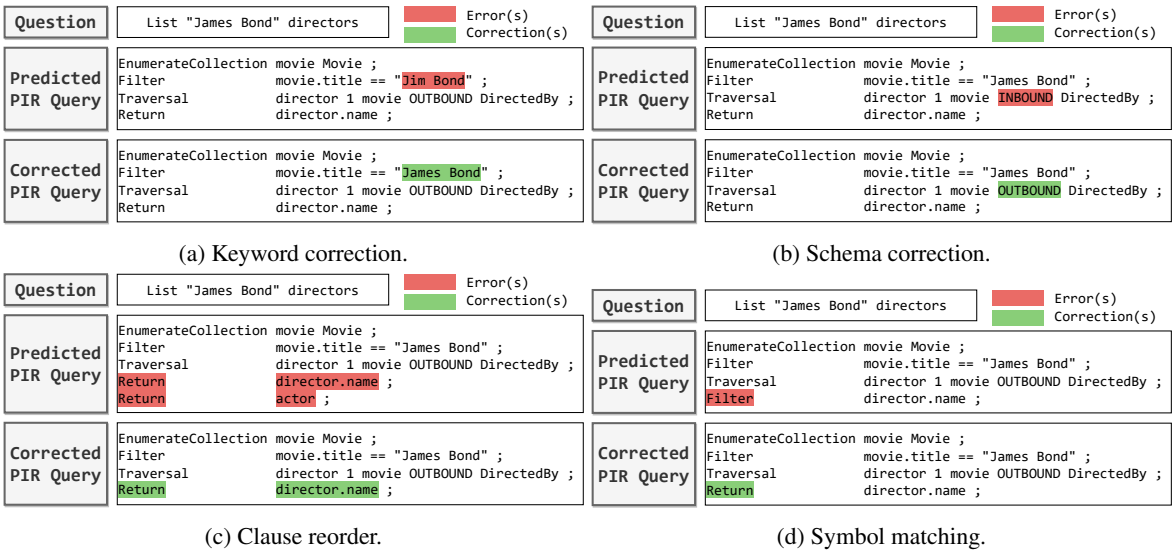
Proof. Consider the first node with exactly one child, denoted as PT_i. The node PT_{i-1} is uniquely identified as the child of PT_i. By removing PT_{i-1} from PT and assigning the number of children for PT_i as $PT_{i-1}.m$, a new sequence with length $N - 1$ is obtained. This procedure can be repeated at most $N - 1$ times until all nodes which have one child are eliminated. The resulting sequence can further uniquely determine a $(0, 2)$-tree according to Theorem 1. □

Algorithm 2 shows the error-tolerant backward compilation procedure. In the error correction step (Lines 1-10), variable u is assigned to indicate whether the predicted MIR query $\hat{r}_{\mathcal{P}}$ is modified in one round of correction (Line 2). In each round, correction rules are repetitively applied to $\hat{r}_{\mathcal{P}}$ (Lines 3-8). The correction procedure ends when no more modifications of $\hat{r}_{\mathcal{P}}$ are detected (Line 9). In the backward compilation step (Lines 13-41), we first create a children placeholder for each node o according to its children number (Lines 11-14). We set N as the length of $\hat{r}_{\mathcal{P}}$, $N_{visited} = 0$ as the number of visited nodes, and $visited$ an array denoting whether a node is visited (Lines 15-16). For each node $o = \hat{r}_{\mathcal{P}}[i]$ whose children number $o.m = 1$ (Lines 17-19), its child is marked visited (Lines 20-21). $o.m$ is then assign with the child number of its child (Line 22). As the construction of a $(0, 2)$-tree according to Theorem 1 (Lines 25-39), we first set $s = 0$ as the number of consecutive leaf nodes in current loop (Line 26). We then enumerate each unvisited node (Lines 27-29), accumulating s by 1 if o is a leaf node (Line 30). When

Algorithm 2: Error-Tolerant Backward Compilation

Input: Target multi-model database $D_{\mathcal{P}}$, generated MIR query $\hat{r}_{\mathcal{P}}$, and correction rule set L.
Output: Predicted MMQL query $\hat{y}_{\mathcal{P}}$.

```
// Error correction
1  while true do
2      u ← false ;
3      for rule in L do
4          r̂'_P ← ApplyRule(rule, r̂_P) ;
5          if r̂'_P ≠ r̂_P then
6              u ← true , r̂_P ← r̂'_P , break;
7          end
8      end
9      if u = false then break;
10 end
   // Backward compilation
11 for o in r̂_P do
12     if o.m = 1 then o.children ← null;
13     if o.m = 2 then o.children ← (null, null);
14 end
15 N ← |r̂_P|, N_visited ← 0 ;
16 visited ← array with size N filled with false;
17 for i = 1 to N do
18     o ← r̂_P[i] ;
19     if o.m = 1 then
20         o.children ← r̂_P[i − 1] ;
21         visited[i − 1] ← true, N_visited ← N_visited + 1
             ;
22         o.m ← r̂_P[i − 1].m ;
23     end
24 end
25 while N_visited < N − 1 do
26     s ← 0 ;          // Number of consecutive leaf nodes
27     for i = 1 to N do
28         if visited[i] = true then continue;
29         o ← r̂_P[i] ;
30         if o.m = 0 then s ← s + 1 ;
31         else
32             tail ← recursively find the tail child from o ;
33             tail.children ← (r̂_P[i − 1], r̂_P[i − 2]) ;
34             visited[i − 1] ← true,
                 visited[i − 2] ← true,
                 N_visited ← N_visited + 2 ;
35             o.m = 0 ;
36             s ← 0 ;
37         end
38     end
39 end
40 ŷ_P ← D_P.synthesize(r̂_P[N − 1]) ;
41 return ŷ_P ;
```



(a) Keyword correction.

(b) Schema correction.

(c) Clause reorder.

(d) Symbol matching.

Figure 6: Illustration of the cases with respect to different error correction rules.

an internal node $o = \hat{r}_{\mathcal{P}}[i]$ is detected, its children are recognized as $(\hat{r}_{\mathcal{P}}[i-1], \hat{r}_{\mathcal{P}}[i-2])$ and merged into the last child node from o (Lines 31-33). These children nodes are marked visited and o is marked as a leaf node (Lines 34-35). Besides, s is reset to zero (Line 36). This procedure is repeated until all nodes except the last root $\hat{r}_{\mathcal{P}}[N-1]$ node are visited. Node $\hat{r}_{\mathcal{P}}[N-1]$ is the root of a reconstructed logical query plan and can be converted to an MMQL query $\hat{y}_{\mathcal{P}}$ by the multi-model database (Line 40).

Figure 5c illustrates the backward compilation of the MIR query in Figure 5b. In the first step (reconstruction from $(0, 1, 2)$-tree), all children of the operators whose children number is equal to 1 are visited (operators 1, 2, 4, 6, and 7). In the second step (reconstruction from $(0, 2)$-tree), two consecutive leaf nodes (operators 3 and 5) are recognized and merged into the children of operator 6. The output logical query plan preserves the same structure with the original logical query plan shown in Figure 5a.

B.2.1 Time Complexity

The time complexity of the error correction step is $O(|L|^2 W^2 H)$, where $|L|$ denotes the amount of correction rules, W denotes the largest word length, and H denotes the largest predicted token length. The time complexity of the backward compilation step is $O(N^2)$ where N is the number of operators in the predicted MIR query.

C Supplementary Experiments

C.1 Experimental Settings

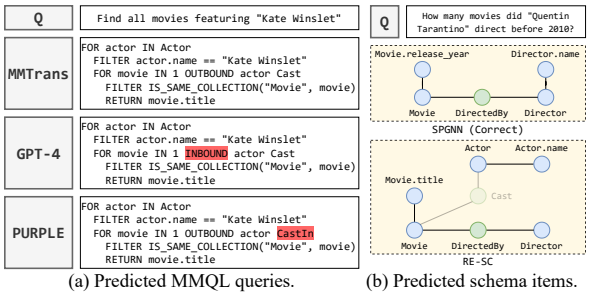
Implementation details. MMTrans is implemented based on PyTorch v2.4.1, in Python v3.9.20 environment. We implement both the base version and the large version of T5 as the PLM of MMTrans. In training the PLM, we adopt Adafactor optimizer with learning rate set to 0.0001. We set the batch size to 8 and searching beam number to 4. In the training of the multi-model schema embedding module, we set learning rate to 0.0001 and node embedding dimension to 768.

In the construction of LLM prompts, we incorporate necessary grammar information, the multi-model schema, and the input question as the prompts in terms of GPT-4/GPT-4o. PURPLE selects relevant SQL queries according to the predicted schema items, which are not directly applicable to the multi-model schema of the proposed dataset. For a fair comparison, for each MMQL query y in the validation dataset, we manually assign the training queries which are determined as the most relevant to y according to PURPLE's classification strategy as the demonstrations.

C.2 Case Study

In this subsection, we discuss several cases to show the effectiveness of the proposed components in MMTrans.

MIR. Figure 6 illustrates cases that the proposed ETBC strategy corrects the wrong predicted MIR queries.



(a) Predicted MMQL queries. (b) Predicted schema items.

Figure 7: Case study results.

Keyword Correction. Figure 6a illustrates the KC case, where the measured similarity between the predicted string value and the keyword in the question is calculated as $sim(v, w) = 1 - 3/(\max(8, 9)) = 0.7 > \theta_{\text{q}}$, where $\theta_{\text{q}} = 0.5$.

Schema Correction. As shown in Figure 6b, the predicted MIR query contains a `Traversal` clause that specifies wrong direction of the edge collection `DirectedBy` from `Movie` vertices to `Director` vertices, since the multi-model schema indicates that only edges with one direction exist. SC rules modify the edge direction to obtain a semantically correct `Traversal` clause. The original predicted MIR query is also syntactically valid and cannot be recognized by the execution guidance strategy, which is widely used in Text-to-SQL tasks.

Clause Reorder. As shown in Figure 6c, the predicted MIR query, containing two consecutive `Return` clauses, would not be compiled to a valid MMQL query. CR strategy analyzes the operands of each `Return` clause and removes the clause with invalid fields. In this example, the second `Return` clause is removed under CR rules, since its operand `actor` cannot be recognized among the former clauses.

Symbol Matching. As shown in Figure 6d, the last clause of the predicted MIR query is a `Filter` clause. However, its operand conforms to the template of a `Return` clause rather than a `Filter` clause. SM rules correct the predicted operator name back to `Return`.

ETBC. Figure 7a illustrates a case where MMTrans correctly predicts the gold AQL query, while GPT-4 and PURPLE-GPT4o both predict with errors. Specifically, GPT-4 predicts the wrong direction of the edge collection `Cast` and PURPLE predicts the wrong name of the edge collection `Cast`, showing that LLMs face challenges to precisely learn the multi-model schema from the input prompt. In contrast, MMTrans corrects these errors during ETBC.

Schema item prediction. Figure 7b displays the schema items predicted by SPGNN and RE-SC. Benefiting from the path augmentation, SPGNN predicts schema items as SPG nodes that form a connected sub-graph. However, RE-SC only considers relationships between tables and columns, while fails to predict the item `Cast`, which represents an edge class in the multi-model schema.