Multi-Modal Data Exploration via Language Agents

Anonymous ACL submission

Abstract

International enterprises, organizations, and hospitals collect large amounts of multi-modal data stored in databases, text documents, images, and videos. While there has been recent progress in the separate fields of multi-modal data exploration as well as in database systems that automatically translate natural language questions to database query languages, the research challenge of querying both structured databases and unstructured modalities (e.g., texts, images) in natural language remains largely unexplored. In this paper, we propose M^{2}EX [1]—a system that enables multi-modal data exploration via language agents. Our approach is based on the following research contributions: (1) Our system is inspired by a real-world use case that enables users to explore multi-modal information systems. (2) M^{2}EX leverages an LLM-based agentic AI framework to decompose a natural language question into subtasks such as text-to-SQL generation and image analysis and to orchestrate modality-specific experts in an efficient query plan. (3) Experimental results on multi-modal datasets, encompassing relational data, text, and images, demonstrate that our system outperforms state-of-the-art multi-modal exploration systems, excelling in both accuracy and various performance metrics, including query latency, API costs, and planning efficiency, thanks to the more effective utilization of the reasoning capabilities of LLMs.

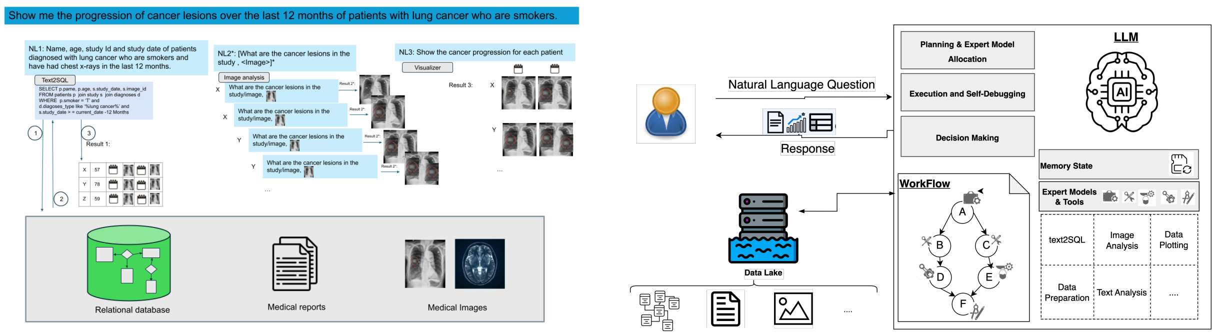
1 Introduction

The rapid expansion of multi-modal data; spanning structured tables, text, images, and video; has created an urgent need for flexible, scalable systems for complex data exploration. In fields like healthcare, users often query across EHRs, medical images, and clinical notes using natural language. However, current systems struggle with integrating modalities, capturing user intent, and optimizing execution workflows, limiting their real-world utility. Traditional solutions focus on single-modality tasks such as text-to-SQL (Sivasubramaniam et al., 2024; Nooralahzadeh et al., 2024; Pourreza and Rafiei, 2024), visual question answering (Li et al., 2023a; Ko et al., 2023; Du et al., 2023), or domain-specific QA (Dong et al., 2024; Liu et al., 2024b), often relying on rigid pipelines or handcrafted logic. While effective in narrow settings, they lack the flexibility to handle heterogeneous data or dynamic analytical goals.

Recent large language models (LLMs) and vision-language models (VLLMs) offer broader generalization but remain limited in real-world multimodal use. Techniques like retrieval-augmented generation (RAG) improve grounding but often fail at structured reasoning, long-term context, and precise tool use; especially in domains that require deep alignment across modalities and step-wise execution. Efforts to inspire LLMs with agentic capabilities – such as ReAct (Yao et al., 2023), tool invocation (Yang et al., 2023; Schick et al., 2023), or workflow automation (Liu et al., 2024a; Urban and Binnig, 2024) – have further exposed systemic challenges.

Existing frameworks frequently adopt rigid, sequential decision-making processes, incurring computational overhead and limiting scalability. Evaluations of these systems are often conducted on in-house datasets, lacking rigorous benchmarking against ground-truth metrics or real-world multimodal contexts. Moreover, many approaches enforce fixed task-planning hierarchies or routing mechanisms, stifling adaptability and reusability across diverse applications. This "one-size-fits-all" mentality contrasts starkly with the need for modular, composable agents capable of dynamically integrating domain-specific tools, retaining contextual memory, and self-optimizing workflows.

[1] Data and code repository are available at `https://anonymous.4open.science/r/M2EX-paper-87C0/README.md`.


Show me the progression of cancer lesions over the last 12 months of patients with lung cancer who are smokers.
NL1: Name, age, study Id and study date of patients diagnosed with lung cancer who are smokers and have had chest x-rays in the last 12 months.
Text2SQL
1
2
3
Result 1:
X 57
Y 78
Z 59
NL2*: [What are the cancer lesions in the study , <Image>]*
Image analysis
What are the cancer lesions in the study/image,
Result 2*:
NL3: Show the cancer progression for each patient
Visualizer
Result 3:
Relational database
Medical reports
Medical Images
Natural Language Question
Response
Data Lake
Planning & Expert Model Allocation
Execution and Self-Debugging
Decision Making
LLM
AI
Memory State
Expert Models & Tools
WorkFlow
A
B
C
D
E
F
text2SQL
Image Analysis
Data Plotting
Data Preparation
Text Analysis

```
SELECT p.pame, p.age, s.study_date, s.image_id
FROM patients p  join study s  join diagnoses d
WHERE  p.smoker = 'T' and
d.diagoses_type like '%lung cancer%' and
s.study_date > = current_date -12 Months
```

Figure 1: **(Left)**: Example workflows of multi-modal data exploration in natural language over heterogeneous data sources. **(Right)**: M^{2}EX system architecture.

To understand these challenges, a concrete scenario of **multi-modal exploration** involving a relational database, text documents, and images is outlined here. A seemingly straightforward query like *Show me the progression of cancer lesions over the last 12 months of patients with lung cancer who are smokers* (see Figure 1, Left) requires multi-modal integration, posing challenges in decomposition and optimization. Critical to this process is optimizing the workflow sequence, i.e., determining which queries should be executed first to minimize computational overhead and maximize efficiency.

In this work, we propose a novel framework for multi-modal data exploration that bridges these gaps through LLM-based agents designed for extensibility, precision, and cross-domain generalization. Our approach combines a "Swiss army knife" philosophy — enabling reusable, adaptable modules for tasks like semantic parsing, cross-modal retrieval, and structured data operations — with a principled evaluation strategy spanning diverse benchmark datasets. By decoupling task planning from execution and incorporating feedback-driven memory, our system supports iterative exploration while mitigating the pitfalls of shallow evaluation and fixed workflows. We demonstrate its efficacy across text, visual, tabular, and hybrid data domains, underscoring the potential of agentic LLMs to unify multi-modal analysis in a scalable, user-centric paradigm.

- *Heterogeneous data understanding:* How can we accurately interpret natural language queries over diverse data types such as text, tables, and images?
- *Workflow orchestration:* How can we decompose a complex query into sub-tasks, organize them into an executable workflow, and delegate each to the right model or tool; respecting dependencies and enabling parallelism?
- *Explainability:* How can we provide users with traceable, transparent results, showing how answers were derived, what data contributed, and where uncertainty remains?

In this paper, we propose M^{2}EX—a multi-modal data exploration system that uses a *LLM-based agentic framework* to tackle these challenges. The basic idea is to first decompose a complex natural language question into simpler sub-questions. Each sub-question is then translated into a workflow of specific tasks. By applying *smart planning*, our approach can reason about which task in the workflow fails and thus re-plan that specific task rather than restarting the complete workflow. The advantage of our approach compared to similar systems such as CAESURA (Urban and Binnig, 2024) is that it enables *parallel task execution* through the construction of a directed acyclic task graph and requires a lower number of tokens from prompt engineering, resulting in more efficient query execution times and API calling costs.

The main contributions of our paper are as follows: (i) **Unified DAG–first planning.** The planner compiles a natural-language query directly into an execution *directed-acyclic graph (DAG)*; independent subtasks therefore run in parallel without a second "physical-plan" stage. (ii) **Self-debug & selective re-planning for speed.** Each expert tool validates its own output once; if a fault persists, the agent rewires *only the affected sub-graph*. This cuts end-to-end latency by up to 51% and reduces token usage by 18% on the ArtWork benchmark. (iii) **Zero-shot cross-domain generalisation.** With a single prompt set and no in-context examples, M^{2}EX attains up to 42% higher answer accuracy than CAESURA and NeuralSQL on ArtWork, RotoWire, and EHRXQA.

2 Related Work

Text-to-SQL systems. The research field of text-to-SQL systems has seen tremendous progress over the last few years (Floratou et al., 2024; Pourreza and Rafiei, 2024) due to advances in large language models. Original success can be attributed to rather simplistic datasets consisting of databases with only several tables, as in Spider (Yu et al., 2018). Especially the introduction of new benchmarks such as ScienceBenchmark (Zhang et al., 2024b), FootbalDB (Fürst et al., 2024), BIRD (Li et al., 2024b) or SM3 (Sivasubramaniam et al., 2024) has further pushed the limits of these systems. Most of the research efforts have been restricted to querying databases in English apart from a few exceptions such as Statbot.Swiss (Nooralahzadeh et al., 2024).

Multi-modal systems. Video Database Management Systems (VDBMSs) support efficient and complex queries over video data, but are often restricted to videos only (e.g., Zhang et al., 2023; Kang et al., 2019; Kakkar et al., 2023). ThalamusDB (Jo and Trummer, 2024) enables queries over multi-modal data but requires SQL as input, with explicit identification of the predicates that should be applied to an attribute corresponding to video or audio data. Similarly, MindsDB[2] and VIVA (Kang et al., 2022) require that users write SQL and manually combine data from relational tables and models. Vision-language models provide textual descriptions of video data (Zhang et al., 2024a), but are not designed to support precise, structured queries. Recent multi-modal systems such as MAGMA (Doe et al., 2025), and LLaVA-Next (Li et al., 2024a) extend vision-language reasoning via unified interfaces or tool-based controllers. However, these models are largely limited to vision-only pipelines and lack support for structured tool orchestration across modalities. In contrast, M^{2}EX generalizes to diverse tool types—including text-to-SQL, Python plotting, and image-VQA—via explicit DAG planning and partial re-planning, enabling scalable and interpretable execution across multi-modal queries.

Closest to our work are CAESURA (Urban and Binnig, 2024), PALIMPZEST (Liu et al., 2024a), and MAT (Gao et al., 2025), which address multi-modal querying and AI workload optimization. In contrast, M^{2}EX focuses on efficient orchestration of model calls and dependencies, reducing latency and cost while improving accuracy by minimizing interference from intermediate outputs (Schick et al., 2023)[3].

[2]https://docs.mindsdb.com

While related systems emphasize query planning, they fall short in enhancing the *accuracy* and *explainability* of model outputs—critical needs in domains like medical data science, where regulatory standards require transparent and justifiable results.

3 Method and System Design

Problem statement. Given a multi-modal query q, a data lake D, a tool catalogue T with metadata T_{meta}, our goal is to produce a directed acyclic task graph $G = (V, E)$ and a final answer a such that each node $v \in V$ is a *(tool, args)* pair, edges E encode data dependencies, the execution of G is valid w.r.t. T_{meta}, and a maximizes task-level answer accuracy.

Proposed System: To address this problem, M^{2}EX enables multi-modal data exploration via language agents. Its details are presented in Algorithm 1 and Figure 1 (right) (A fully annotated DAG and end-to-end use-case example appear in Figures 2 and 3). M^{2}EX is an *agentic system* (Kapoor et al., 2024) driven by LLMCompiler (Kim et al., 2023), a dynamic planner pattern based on a Large Language Model, equipped with a comprehensive toolkit $\mathcal{T}$ containing all the necessary models to decompose a user's request, such as a multi-modal natural language question, into a workflow (i.e., a graph of sub-questions). The workflow is represented as a Directed Acyclic Graph (*DAG*), where each node corresponds to a simple sub-task (or sub-question) with a specific tool assigned by the planner. While decoupling logical and physical plans can be suboptimal due to plan ambiguity and nonlinearity, unlike CAESURA, the planner determines sub-tasks that can be executed in parallel and manages their dependencies by leveraging an LLM to directly generate the execution plan from the query as a graph of function calls. M^{2}EX is designed to be adaptable, allowing dynamic debugging and plan modification (re-planning) when necessary, for example, if a failure occurs during a text-to-SQL sub-task.

As shown in Algorithm 1 and Figure 1, the system is composed of the following key components: (1) *User Query (q):* a multi-modal natural language question posed by the user, which initiates

[3]CAESURA and MAT employ the ReAct agent framework, which leads to extended context tokens and increased latency.

Algorithm 1 M^{2}EX: Multi-Modal Data Exploration via Language Agents

```
Require: User query $q$, Agent Core $\mathcal{LLM}$, toolkit $\mathcal{T}$, Data Lake $D$, Pre-
    defined Prompts $\mathcal{P}$, Empty memory state $\mathcal{R}$
Ensure: Final answer $a$
 1 Stage 1: Planning & Expert Model Allocation
 2 $\mathcal{R} \leftarrow \mathcal{R} \cup \{q, D_{meta}\}$
 3 $S \leftarrow$ DECOMPOSE($\mathcal{R}, \mathcal{LLM}, \mathcal{T}_{meta}$)  ▷ Use an agent core $\mathcal{LLM}$
   (with a planner prompt $\in \mathcal{P}$ access to tool metadata) to decompose $q$
   into subtasks $s_1, \ldots, s_n$. Each task contains a tool, arguments, and list
   of dependencies.
 4 $G \leftarrow$ BUILDDAG($\mathcal{S}, \mathcal{LLM}$)  ▷ Construct a
   Directed Acyclic Graph (DAG): $G$ where each node represents a subtask
   and edges represent dependencies
 5 Stage 2: Execution & Self-debugging
 6 $\sigma \leftarrow$ TOPOLOGICALSORT($G$)  ▷ Determine an execution order that
   respects dependencies
 7 $\mathcal{B} \leftarrow$ GROUPPARALLELTASKS($\sigma, G$)  ▷ Partition tasks into parallel
   execution
 8 for each batch $b_k \in \mathcal{B}$ do
 9     Launch parallel execution:
10     for each subtask $s_i \in b_k$ do
11         $r_i \leftarrow$ EXECUTE($s_i, \mathcal{T}, \mathcal{D}$)  ▷ Invoke the assigned expert
   tool for $s_i$. Integrate $n$-time self-debugging to automatically detect and
   correct errors as needed. ($n = 1$). If there is still an error, provide an
   error message as an output of execution.
12         $\mathcal{R} \leftarrow \mathcal{R} \cup \{r_i\}$
13     end for
14 end for
15 Stage 3: Decision Making
16 Validate $\mathcal{R}$ via reflection  ▷ Check that outputs are correct and
   executable; if not, trigger error feedback.
17 if validation fails then
18     $G \leftarrow$ REPLAN($G, \mathcal{R}, \mathcal{LLM}, \mathcal{T}_{meta}$)  ▷ Dynamically adjust
   the DAG (e.g., reallocate tasks or update tool parameters) based on error
   feedback using an agent core $\mathcal{LLM}$ (with a replanning prompt $\in \mathcal{P}$).
19     goto line 5  ▷ Restart execution with the updated plan.
20 end if
21 $a \leftarrow$ SYNTHESIZE($\mathcal{R}, \mathcal{LLM}$)  ▷ Aggregate and refine intermediate
   results into the final answer using LLM reasoning.
22 if $a$ is insufficient or uncertain then
23     $G \leftarrow$ REPLAN($G, \mathcal{R}, \mathcal{LLM}, \mathcal{T}_{meta}$)  ▷ Dynamically adjust
   the DAG (e.g., reallocate tasks or update tool parameters) based on error
   feedback using an agent core $\mathcal{LLM}$ (with a replanning prompt $\in \mathcal{P}$).
24     goto line 5  ▷ Restart execution with the updated plan.
25 end if
26 return $a$
```

the process of task decomposition and execution. (2) *Agent Core ($\mathcal{LLM}$):* the core reasoning engine that powers the dynamic planning, execution, and decision-making processes. The LLM is responsible for decomposing the user query into subtasks, managing dependencies, and synthesizing final results using diverse prompts $\mathcal{P}$. (3) *Expert Models & Tools (Toolkit) ($\mathcal{T}$):* a comprehensive collection of expert models and tools that are used for executing specific sub-tasks. The toolkit provides the necessary models for tasks such as `text-to-SQL`, `text analysis`, `image analysis`, `data preparation`, and `data plotting`. Each expert model or tool should include a description and argument specifications ($\mathcal{T}_{meta}$), and they will be available during the planning and re-planning stages. (4) *Data Lake (D):* a central repository that stores both structured and unstructured data, such as tabular data, images, and text. Each expert model and tool has direct access to the data lake to perform its assigned tasks. The data stored in the lake is utilized as input for various tasks, enabling the system to generate accurate results for the user's query. (5) *Pre-defined Prompts ($\mathcal{P}$):* a collection of predefined prompts available to the LLM, which are used to guide the reasoning process during planning, execution, and decision-making (see details in Appendix B). (6) *Memory State ($\mathcal{R}$):* The initial memory state starts empty and captures all intermediate results and interactions throughout the workflow execution. The system tracks these intermediate results using an output object that stores the answer and reasoning at each node in the workflow. (7) *Final Answer (a):* The final answer is the output generated by the system after executing all the tasks and performing reasoning through the LLM. It consolidates all intermediate results and provides a comprehensive response to the user's query. The final answer typically includes several components: a summary of the task or query result, detailed information about the outcome, the source of the data used, an inference indicating the success of the task, and any additional explanations or clarifications. This structured output ensures that the user receives not only the result but also the reasoning and context behind it. In Figure 2, we demonstrate the showcase of M^{2}EX using an example query applied to the EHRXQA data, which includes relational tables and images: *Was patient 18061894 prescribed acetaminophen, and did a chest x-ray show any technical assessments until 12/2103?*

The system starts with the user query q and processes it through several stages, as detailed below: (i) *Planning & Expert Model Allocation.* The system begins by analyzing the user query q and decomposes it into a sequence of tasks. Using the agent core ($\mathcal{LLM}$), the system identifies the required expert models and tools from the toolkit $\mathcal{T}$, along with their input arguments and interdependencies. These subtasks are synthesized into a workflow represented as a Directed Acyclic Graph (*DAG*), G, where each node represents a task, and edges represent dependencies between them. E.g., a natural language question can be split into multiple tasks such as `intent table detection`, `text2SQL`, and `image analysis` as shown in Figure 2. The workflow reflects the execution sequence and dependencies that are necessary to answer the user's query. The system also utilizes predefined prompts $\mathcal{P}$ to guide the reasoning process during task decomposition.

(ii) *Execution and Self-Debugging.* The system executes the tasks according to the generated work-

flow by invoking the relevant expert models and tools from the toolkit $\mathcal{T}$. The system utilizes a state object $\mathcal{R}$, which stores intermediate results and interactions during the execution. The tasks are partitioned into independent batches $\mathcal{B}$ that can be executed in parallel, which is determined through a topological sort (TopologicalSort(G)) of the DAG. For each batch, the system launches parallel executions of the assigned tasks. The tasks are executed using the expert models, and the outcomes are passed on to subsequent tasks that depend on them. Each expert model includes a self-debugging mechanism to detect and correct errors during execution. If an error persists, the system can provide feedback and retry the process, thereby enhancing the robustness of the execution.
(iii) *Decision Making.* After the execution of the subtasks, M^{2}EX inspects the intermediate results stored in $\mathcal{R}$ to determine whether they are sufficient to fulfill the user's request. If the results are satisfactory, the system synthesizes them into the final answer a. However, if the results are insufficient or uncertain, the system triggers a re-planning process by invoking Replan($G, \mathcal{R}, \mathcal{LLM}, \mathcal{T}_{\text{meta}}$) to adjust the DAG and re-execute the tasks. This process repeats until the decision-making component is satisfied with the final result or a predefined maximum loop limit is reached.

In summary, M^{2}EX uses an algorithmic approach where the system first decomposes the user query into subtasks, executes these tasks with error detection and correction mechanisms, and synthesizes the results into a final answer. The system is highly adaptive, with dynamic re-planning capabilities powered by the reasoning abilities of the LLM to ensure efficient task execution, debugging, and modification of the plan when needed. Our current M^{2}EX implementation offers a range of features, including self-debugging, query re-planning, optimization, and explainability to better understand how a natural language question is decomposed into multiple sub-tasks. See details in Appendix D.

Complexity and convergence. Planning inspects $|S|$ subtasks and calls Φ once, costing $\mathcal{O}(|S|\, C_{\text{LLM}})$ tokens (C_{LLM} = context length processed by the language model). With unlimited workers, execution latency is $\mathcal{O}(\text{depth}(G))$; with p workers it is bounded by depth(G) as well. Re-planning only touches the affected sub-DAG, so its worst-case cost is strictly $\leq$ the first planning pass.

4 Experiments

In this section, we evaluate M^{2}EX's performance, focusing on the following research questions: (1) How well does M^{2}EX tackle multi-modal natural language questions on three different datasets consisting of tabular data and images? (2) How does the system perform compared to state-of-the-art systems such as CAESURA (Urban and Binnig, 2024) and NeuralSQL (Bae et al., 2024) on underlying benchmark datasets? (3) What systematic errors can we observe?

4.1 Experimental Setup

Datasets For our experiments, we used three different datasets, namely datasets about artwork, basketball, as well as electronic health records. Due to hardware limitations, we reduced the dataset to 100 images and reports. Processing the full size in CAESURA can result in crashes due to out-of-memory issues.
Dataset 1: ArtWork. This dataset was introduced by Urban and Binnig (2024) and contains information about paintings in tabular form as well as an image collection containing 100 images of the artworks, collected from Wikipedia. The tabular data contains metadata about paintings such as title, inception, movement, etc. as well as a reference to the respective paintings. A typical example question from this dataset is *Plot the number of paintings depicting war for each century* (see Figure 3 in the Appendix).
In addition to the 24 existing questions in the ArtWork dataset, we propose six new questions aimed at evaluating parallel task planning and execution, facilitating a comparison between the characteristics of the two architectures. These six questions incorporate both single and multiple modalities. Moreover, four of the six questions require responses in various formats: two questions demand two plots, and two questions involve a combination of plotting and showing the results in a specific data structure, i.e. either as a tabular format or as a JSON format. The final test dataset contains 30 natural language questions derived from the original 24 in the ArtWork dataset. These include 8 queries seeking a single result value, 11 requiring structured data as output, and 11 requesting a plot. Of these, 18 queries involve multi-modal data, while the remaining 12 are based exclusively on relational data. We have chosen this dataset to directly compare our system with CAESURA (Ur-

ban and Binnig, 2024), one of the state-of-the-art systems for multi-modal data exploration in natural language.

Dataset 2: RotoWire. This dataset is also utilized by Urban and Binnig (2024) and consists of one relational database and 100 randomly selected textual reports about NBA games, including metadata, key statistics of individual players, and team performance metrics. A typical example question from this dataset is *Plot the highest number of three-pointers made by players from each nationality*. The test dataset comprises 12 natural language questions, evenly divided into 6 single-modal and 6 multi-modal queries. Regarding output format, 3 questions require a single value as a response, 5 involve structured data outputs, and 4 necessitate visualization through plots.

Dataset 3: Electronic Health Records (EHR). We also utilized the EHRXQA (Bae et al., 2024) dataset, a multi-modal question answering dataset that integrates structured electronic health records (EHRs) with chest X-ray images. This dataset consists of 18 tables and 432 images, and specifically requires cross-modal reasoning. The questions of EHRXQA are categorized based on their scope in terms of modality and patient relevance. For *modality-based* categorization, questions were classified into three types: Table-related, image-related, and table-image-related, based on the data modality required. The *patient-based* categorization classified questions based on their relevance to a single patient, a group of patients, or none (i.e., unrelated to specific patients). We have chosen this dataset since it was used to evaluate NeuralSQL, another state-of-the-art system for multi-modal data exploration. To manage the cost of an API call, we extracted 100 questions randomly. The selection process was guided by three predefined categories within the test set of the EHRXQA dataset: Image Single-1, Image Single-2, and Image+Table Single (for details, please look at Bae et al. (2024)).

Several considerations influenced our decision to work with reduced versions of these datasets: *Demonstrating Viability* The reduced dataset size demonstrates M^{2}EX's viability across diverse multi-modal datasets with ground truth, proving its ability to handle complex queries in a controlled setting. *Complexity of Building Datasets* Constructing large-scale multi-modal datasets with precise ground truth is a complex, manual process, which limits the scaling-up within the study's scope. *Cost Considerations* The cost of API calls to the LLM powering M^{2}EX necessitates a balance between dataset size and experimental feasibility, ensuring thorough evaluation within practical constraints.

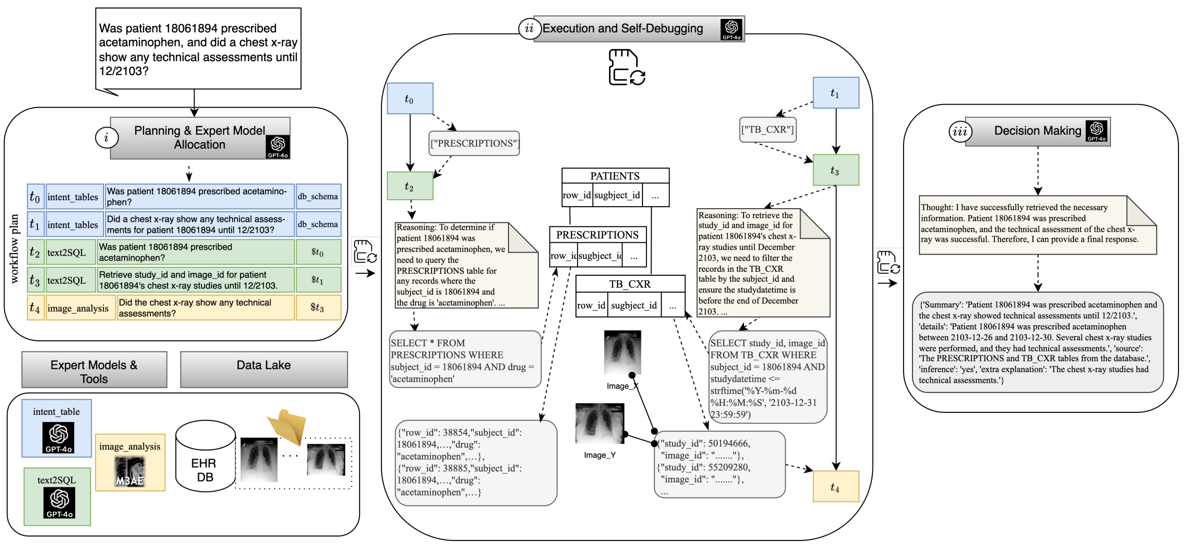
4.2 Baseline Systems and Setup

We compare M^{2}EX to the baseline implementations of CAESURA (Urban and Binnig, 2024) and NeuralSQL (Bae et al., 2024) - two important state-of-the-art systems for multi-modal data exploration.

CAESURA supports natural language queries over a multi-modal data lake, leveraging BLIP-2 (Li et al., 2023b) for visual question answering and a fine-tuned BART (Lewis et al., 2020) for text question answering. We reproduced the results of CAESURA on the ArtWork and RotoWire datasets using GPT-4o for planning, data processing, and plot generation while adopting the other tool models as proposed in CAESURA (Urban and Binnig, 2024). For comparison with our system, we use GPT-4o as the LLM for both planning and text analysis on RotoWire. On ArtWork, we employ GPT-4o as the planner and retain the same model for visual question answering (i.e., BLIP-2) in M^{2}EX.

In NeuralSQL, an LLM is integrated with an external visual question answering system, M3AE model (Chen et al., 2022), to handle multi-modal questions over a structured database with images by translating a user question to SQL in one step. To ensure that we used the optimal hyperparameter settings and prompt structure, we contacted the authors of EHRXQA (Bae et al., 2024), who provided the results of their experiment for NeuralSQL using GPT-4o on 100 randomly selected questions.

For M^{2}EX, we employ the M3AE model with task-specific fine-tuned weights, provided by (Bae et al., 2024), for the image analysis task. The customized M3AE model is encapsulated as a web service and deployed on the same computing node as our experiments. We conduct the experiments using a CUDA-accelerated computational node on an OpenStack virtual host. This node is equipped with a 16-core CPU, 16 GB of main memory, and 240 GB of SSD storage. Additionally, it features an NVIDIA T4 GPU with 16 GB of dedicated graphics memory. A complete mapping of subtasks to expert models/tools and prompt types is provided in Table 4 (Appendix C).



Was patient 18061894 prescribed acetaminophen, and did a chest x-ray show any technical assessments until 12/2103?

t_0	intent_tables	Was patient 18061894 prescribed acetamino-phen?	db_schema
t_1	intent_tables	Did a chest x-ray show any technical assess-ments for patient 18061894 until 12/2103?	db_schema
t_2	text2SQL	Was patient 18061894 prescribed acetaminophen?	$\$t_0$
t_3	text2SQL	Retrieve study_id and image_id for patient 18061894's chest x-ray studies until 12/2103.	$\$t_1$
t_4	image_analysis	Did the chest x-ray show any technical assessments?	$\$t_3$

Reasoning: To determine if patient 18061894 was prescribed acetaminophen, we need to query the PRESCRIPTIONS table for any records where the subject_id is 18061894 and the drug is 'acetaminophen'. ...

```
SELECT * FROM
PRESCRIPTIONS WHERE
subject_id = 18061894 AND drug =
'acetaminophen'
```

```
{"row_id": 38854,"subject_id":
18061894,…,"drug":
"acetaminophen",…},
{"row_id": 38885,"subject_id":
18061894,…,"drug":
"acetaminophen",…}
```

Reasoning: To retrieve the study_id and image_id for patient 18061894's chest x-ray studies until December 2103, we need to filter the records in the TB_CXR table by the subject_id and ensure the studydatetime is before the end of December 2103. ...

```
SELECT study_id, image_id
FROM TB_CXR WHERE
subject_id = 18061894 AND
studydatetime <=
strftime('%Y-%m-%d
%H:%M:%S', '2103-12-31
23:59:59')
```

```
{"study_id": 50194666,
 "image_id": "......."},
{"study_id": 55209280,
 "image_id": "......."},
...
```

Thought: I have successfully retrieved the necessary information. Patient 18061894 was prescribed acetaminophen, and the technical assessment of the chest x-ray was successful. Therefore, I can provide a final response.

```
{'Summary': 'Patient 18061894 was prescribed acetaminophen and the chest x-ray showed technical assessments until 12/2103.', 'details': 'Patient 18061894 was prescribed acetaminophen between 2103-12-26 and 2103-12-30. Several chest x-ray studies were performed, and they had technical assessments.', 'source': 'The PRESCRIPTIONS and TB_CXR tables from the database.', 'inference': 'yes', 'extra explanation': 'The chest x-ray studies had technical assessments.'}
```

Figure 2: M^{2}EX system architecture in EHRXQA (Bae et al., 2024) with an example of processing a multi-modal query. The query is automatically decomposed into various components which can be inspected by the user for explainability.

4.3 Evaluation Metrics

To evaluate M^{2}EX against state-of-the-art systems, we use the following metrics: (i) *Accuracy*: Measures the accuracy (i.e., exact match) of the generated result set compared with the gold standard result set or with the human expert. (ii) *Steps*: Number of steps required by the respective system to come up with the final result. These steps include reasoning, planning, re-planning, etc. (iii) *Tokens*: Number of tokens used for prompt engineering. (iv) *Latency*: End-to-end execution time for a system to come up with the final result. (v) *API costs*: Costs for calling the LLM, e.g. for GPT4o.

We apply the above-mentioned metrics under various questions and system categories:

(i) *Modality*: Questions can either be of *single* modality, i.e., querying only relational data or image data, or of *multiple* modalities, i.e., querying both relational and image data. (ii) *Output Type*: The output type of a question can either be a *single value*, e.g., true or false, a *data structure*, e.g., in tabular or JSON format, a plot, or a combination of plots and data structures. (iii) *Workflow*: The generated workflow plan can either be *sequential* or *parallel*. Finally, we evaluate if a system generates a correct (multi-modal) query plan (i.e., generated plan), and if it supports re-planning.

4.4 Results on the Benchmark Datasets

Results on the ArtWork and RotoWire Datasets Table 1 shows M^{2}EX outperforms CAESURA by 30% on the ArtWork and by ca. 42% on the RotoWire datasets in accuracy, with advantages in both single- and multi-modality queries. Efficiency-wise, M^{2}EX excels on ArtWork with fewer steps, lower latency, and reduced costs. On RotoWire, despite higher token usage and costs due to advanced text analysis, M^{2}EX maintains superior accuracy. Additionally, M^{2}EX supports re-planning and offers better explanations, features absent in CAESURA.

Results on the EHRXQA Dataset In Table 2, M^{2}EX outperforms NeuralSQL in overall accuracy (51.00% vs. 33.00% in 10-shot) on the EHRXQA dataset, especially in multiple-table queries (77.50% vs. 47.50%) and binary questions (74.00% vs. 48.00%). Additionally, M^{2}EX provides plan generation (98% coverage), explanations, and replanning—features that NeuralSQL lacks. Metrics like steps, tokens, and latency are excluded since NeuralSQL generates answers directly without intermediate steps, unlike M^{2}EX's transparent workflow. We exclude CAESURA from the EHRXQA experiments due to its inefficiency with EHRXQA's complex schema. While CAESURA is intended to be a general-purpose multi-modal system, it processes the relational database through multiple steps, examining each table and relationship sequentially. This limitation introduces significant overhead when handling the complex data schema of the EHRXQA dataset (there are 18 tables) during its discovery phase. Consequently, reproducing CAESURA on EHRXQA questions fails to perform inferences at the early stages of

System	Category (# in ArtWork|# in RotoWire)		ArtWork						RotoWire						Re-planning
			Accuracy	Steps	Tokens	Latency [s]	Cost [$]	Gen. Plan	Accuracy	Steps	Tokens	Latency [s]	Cost [$]	Gen. Plan	
CAESURA few-shot (4) in planning	**Modality**	Single (15|6)	60.00%	152	214,014	973.28	1.33	80%	50.00%	79	100,277	500.52	0.65	91.67%	No
		Multiple (15|6)	6.67%	164	268,918	4,847.95	1.65		0.00%	78	133,230	959.17	0.85		
	Output Type	Single Value (8|3)	37.50%	88	135,077	1,047.24	0.82		66.67%	32	45,145	287.55	0.29		
		Data Structure (10|5)	50.00%	116	183,454	2,683.03	1.14		20.00%	69	104,345	659.37	0.68		
		Plot (8|4)	25.00%	79	112,732	1,856.66	0.69		0.00%	56	84,017	512.77	0.53		
		Plot-Plot (2|0)	0%	16	21,508	108.87	0.14		–	–	–	–	–		
		Plot-Data Structure (2|0)	0%	17	30,161	125.42	0.19		–	–	–	–	–		
	Workflow	Sequential (24|12)	41.67%	261	399,045	5,330.12	2.45		25.00%	157	233,507	1,459.69	1.50		
		Parallel (6|0)	0%	55	83,887	491.11	0.52		–	–	–	–	–		
	Overall (30|12)		33.33%	316	**482,932**	5,821.23	2.98		25.00%	157	**233,507**	**1,459.69**	**1.50**		
M^{2}EX zero-shot	**Modality**	Single (15|6)	100.00%	96	159,212	525.09	0.61	100%	100.00%	34	89,810	524.06	0.40	100%	Yes
		Multiple (15|6)	26.67%	107	326,400	2,515.03	1.49		33.33%	42	952,386	3,235.96	3.22		
	Output Type	Single Value (8|3)	50.00%	56	71,575	494.78	0.39		100.00%	16	108,520	499.70	0.40		
		Data Structure (10|5)	50.00%	67	223,528	1,330.40	0.89		40.00%	27	410,698	2,120.15	1.57		
		Plot (8|4)	75.00%	52	118,431	798.97	0.48		75.00%	33	522,987	1,140.17	1.65		
		Plot-Plot (2|0)	100.00%	14	50,108	308.92	0.22		–	–	–	–	–		
		Plot-Data Structure (2|0)	100.00%	14	21,970	107.05	0.10		–	–	–	–	–		
	Workflow	Sequential (24|12)	62.50%	163	338,766	2,131.11	1.51		66.67%	76	1,042,196	3,760.02	3.62		
		Parallel (6|0)	66.67%	40	146,846	909.01	0.59		–	–	–	–	–		
	Overall (30|12)		**63.33%**	**203**	485,612	**3,040.12**	**2.10**		**66.67%**	**76**	1,042,196	3,760.02	3.62		

Table 1: Performance metrics of Caesura (Urban and Binnig, 2024) and M^{2}EX on ArtWork and RotoWire. Planner coverage (*Gen. Plan*) is 100% on ArtWork and RotoWire, indicating reliable task decomposition across domains.

System		Scope			Output Type		Overall (100)	Generated Plan	Replanning
		Image Single-1 (30)	Image Single-2 (30)	Image+Table Single (40)	Binary (50)	Categorical (50)			
NeuralSQL	zero-shot	0.00%	0.00%	0.00%	0.00%	0.00%	0.00%	N/A	No
	few-shot (n = 10)	**26.67%**	20.00%	47.50%	48.00%	18.00%	33.00%		
M^{2}EX	zero-shot	23.33%	**43.33%**	**77.50%**	**74.00%**	**28.00%**	**51.00%**	**98%**	**Yes**

Table 2: Performance metrics of NeuralSQL (zero-shot and few-shot) and M^{2}EX (zero-shot) on EHRXQA. Planner coverage (*Generated Plan*): 98%.

Dataset	System	Tasks	Errors	Dominant Error Source
ArtWork	CAESURA	30	20	Faulty plans; VQA errors
	M^{2}EX	30	11	VQA errors only
RotoWire	CAESURA	12	9	Text analysis; SQL faults
	M^{2}EX	12	4	Text analysis
EHRXQA	NeuralSQL	100	67	N/A – no plan output
	M^{2}EX	100	49	VQA errors only

Table 3: Top-level error breakdown. See App. E for details.

the planning phase, ultimately terminating after exceeding the maximum number of allowed attempts.

4.5 Error Analysis

We evaluate system errors across three datasets: ArtWork, RotoWire, and EHRXQA, identifying key bottlenecks and component failures (see Table 3 and detailed breakdown in Appendix E, Fig. 6). On the ArtWork dataset, CAESURA exhibits 20 errors out of 30 tasks, mainly due to faulty planning in sequential workflows and incorrect outputs from the image analysis module. Multi-modal tasks involving plot and data structure outputs are particularly error-prone, especially in parallel workflows where planning failures are common. By contrast, M^{2}EX achieves full planning success, with image interpretation errors being the only significant issue. In the RotoWire dataset, CAESURA fails on 9 of 12 tasks due to text analysis failures and SQL generation flaws. M^{2}EX resolves all single-modal tasks but faces 4 errors in multi-modal tasks, again tied to text interpretation. These patterns highlight M^{2}EX's robustness in planning and execution while exposing shared weaknesses in text and image understanding across systems.

For the EHRXQA dataset, we focus solely on M^{2}EX due to NeuralSQL's lack of interpretable planning. Of 49 errors, 36 arise in categorical tasks, indicating a strong link between output type and model performance. Most failures originate from inaccurate image analysis by the M3AE model. These results emphasize the need for improved image understanding, especially for categorical reasoning, alongside stronger planning and SQL components. See Appendix E for full error analysis.

5 Conclusions

In this paper, we show that multi-agent collaboration via LLMs (GPT-4o) offers a powerful approach to multi-modal data exploration in natural language. Our system, M^{2}EX, outperforms prior methods across datasets with tabular, text, and image data by leveraging smart re-planning, parallel execution, and transparent, explainable workflows. It blends accuracy, efficiency, and user-centric design, marking a significant advance in multi-modal data exploration, with strong performance in text-to-SQL tasks and potential for further enhancement in image reasoning and workflow optimization.

Future work will focus on better data alignment, prompt design, planning efficiency, and scaling to larger datasets and new modalities such as video and human-in-the-loop interaction.

Limitations

Despite M^{2}EX's overall superior performance, several limitations remain. Most notably, the system's reliance on image analysis introduces a consistent source of error, particularly in tasks involving categorical outputs. The M3AE model often fails to capture subtle visual distinctions, which disproportionately affects the accuracy of multi-modal tasks. We did not explore alternative image processing approaches, as improving the visual pipeline was not the primary objective of this study. Instead, we adopted visual models commonly used in prior work to ensure a fair and consistent basis for comparison. Similarly, we restricted our language model experiments to GPT-4o to both showcase our proposed methods and maintain comparability with recent studies.

Additionally, although M^{2}EX successfully generates plans for all tasks, its performance still hinges on accurate text interpretation. In the RotoWire dataset, for example, errors in multi-modal questions were largely driven by flawed text comprehension, revealing a vulnerability in the language understanding pipeline.

Finally, the system exhibits a performance gap between binary and categorical tasks, suggesting that output type complexity influences success rates. These findings indicate that further improvements are needed in visual reasoning, nuanced language understanding, and output-type generalization.

References

Seongsu Bae, Daeun Kyung, Jaehee Ryu, Eunbyeol Cho, Gyubok Lee, Sunjun Kweon, Jungwoo Oh, Lei Ji, Eric Chang, Tackeun Kim, and 1 others. 2024. Ehrxqa: A multi-modal question answering dataset for electronic health records with chest x-ray images. *Advances in Neural Information Processing Systems*, 36.

Zhihong Chen, Yuhao Du, Jinpeng Hu, Yang Liu, Guanbin Li, Xiang Wan, and Tsung-Hui Chang. 2022. Multi-modal masked autoencoders for medical vision-and-language pretraining. In *Medical Image Computing and Computer Assisted Intervention – MICCAI 2022: 25th International Conference, Singapore, September 18–22, 2022, Proceedings, Part V*, page 679–689, Berlin, Heidelberg. Springer-Verlag.

Jane Doe, John Smith, and Wei Zhang. 2025. Magma: Multi-agent generalist for multimodal alignment. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. To appear.

Junnan Dong, Qinggang Zhang, Chuang Zhou, Hao Chen, Daochen Zha, and Xiao Huang. 2024. Cost-efficient knowledge-based question answering with large language models. In *Advances in Neural Information Processing Systems*, volume 37, pages 115261–115281. Curran Associates, Inc.

Yifan Du, Junyi Li, Tianyi Tang, Wayne Xin Zhao, and Ji-Rong Wen. 2023. Zero-shot visual question answering with language model feedback. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 9268–9281, Toronto, Canada. Association for Computational Linguistics.

Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, Alex van Grootel, Brandon Chow, Kai Deng, Katherine Lin, Marcos Campos, Venkatesh Emani, Vivek Pandit, Victor Shnayder, Wenjing Wang, and Carlo Curino. 2024. Nl2sql is a solved problem... not! In *CIDR*.

Jonathan Fürst, Catherine Kosten, Farhad Nooralahzadeh, Yi Zhang, and Kurt Stockinger. 2024. Evaluating the data model robustness of text-to-sql systems based on real user queries. *arXiv preprint arXiv:2402.08349*.

Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaojian Ma, Tao Yuan, Yue Fan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. 2025. Multi-modal agent tuning: Building a VLM-driven agent for efficient tool usage. In *The Thirteenth International Conference on Learning Representations*.

Saehan Jo and Immanuel Trummer. 2024. Thalamusdb: Approximate query processing on multi-modal data. *Proc. ACM Manag. Data*, 2(3).

Gaurav Tarlok Kakkar, Jiashen Cao, Pramod Chunduri, Zhuangdi Xu, Suryatej Reddy Vyalla, Prashanth Dintyala, Anirudh Prabakaran, Jaeho Bang, Aubhro Sengupta, Kaushik Ravichandran, Ishwarya Sivakumar, Aryan Rajoria, Ashmita Raju, Tushar Aggarwal, Abdullah Shah, Sanjana Garg, Shashank Suman, Myna Prasanna Kalluraya, Subrata Mitra, and 4 others. 2023. Eva: An end-to-end exploratory video analytics system. In *Proceedings of the Seventh Workshop on Data Management for End-to-End Machine Learning*, DEEM '23, New York, NY, USA. Association for Computing Machinery.

Daniel Kang, Peter Bailis, and Matei Zaharia. 2019. Blazeit: Optimizing declarative aggregation and limit queries for neural network-based video analytics. *Proc. VLDB Endow.*, 13(4):533–546.

Daniel Kang, Francisco Romero, Peter D. Bailis, Christos Kozyrakis, and Matei Zaharia. 2022. VIVA: an end-to-end system for interactive video analytics. In *CIDR*.

Sayash Kapoor, Benedikt Stroebl, Zachary S Siegel, Nitya Nadgir, and Arvind Narayanan. 2024. Ai agents that matter. *arXiv preprint arXiv:2407.01502*.

Sehoon Kim, Suhong Moon, Ryan Tabrizi, Nicholas Lee, Michael W Mahoney, Kurt Keutzer, and Amir Gholami. 2023. An llm compiler for parallel function calling. *arXiv preprint arXiv:2312.04511*.

Dohwan Ko, Ji Lee, Woo-Young Kang, Byungseok Roh, and Hyunwoo Kim. 2023. Large language models are temporal and causal reasoners for video question answering. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 4300–4316, Singapore. Association for Computational Linguistics.

Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, page 7871. Association for Computational Linguistics.

Bo Li, Kaichen Zhang, Hao Zhang, Dong Guo, Renrui Zhang, Feng Li, Yuanhan Zhang, Ziwei Liu, and Chunyuan Li. 2024a. Llava-next: Stronger llms supercharge multimodal capabilities in the wild.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *NeurIPS*.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023a. Blip-2: bootstrapping language-image pre-training with frozen image encoders and large language models. In *Proceedings of the 40th International Conference on Machine Learning*, ICML'23. JMLR.org.

Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. 2023b. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR.

Chunwei Liu, Matthew Russo, Michael Cafarella, Lei Cao, Peter Baille Chen, Zui Chen, Michael Franklin, Tim Kraska, Samuel Madden, and Gerardo Vitagliano. 2024a. A declarative system for optimizing ai workloads. *arXiv e-prints*, pages arXiv–2405.

Lihui Liu, Blaine Hill, Boxin Du, Fei Wang, and Hanghang Tong. 2024b. Conversational question answering with language models generated reformulations over knowledge graph. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 839–850, Bangkok, Thailand. Association for Computational Linguistics.

Farhad Nooralahzadeh, Yi Zhang, Ellery Smith, Sabine Maennel, Cyril Matthey-Doret, Raphaël de Fondville, and Kurt Stockinger. 2024. StatBot.Swiss: Bilingual Open Data Exploration in Natural Language. In *Findings of ACL*.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *NeurIPS*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessi, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language Models Can Teach Themselves to Use Tools. *Advances in Neural Information Processing Systems*, 36:68539–68551.

Sithursan Sivasubramaniam, Cedric Osei-Akoto, Yi Zhang, Kurt Stockinger, and Jonathan Fuerst. 2024. SM3-text-to-query: Synthetic multi-model medical text-to-query benchmark. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*.

Matthias Urban and Carsten Binnig. 2024. CAESURA: language models as multi-modal query planners. In *CIDR*.

Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. 2023. Mm-react: Prompting chatgpt for multimodal reasoning and action.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*.

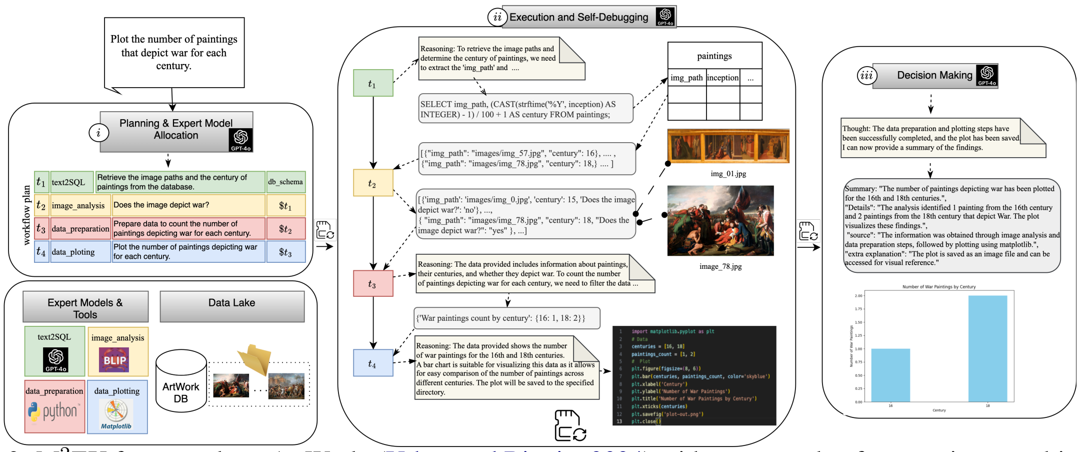
Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Enhao Zhang, Maureen Daum, Dong He, Brandon Haynes, Ranjay Krishna, and Magdalena Balazinska. 2023. Equi-vocal: Synthesizing queries for compositional video events from limited user interactions. *Proceedings of the VLDB Endowment*, 16(11):2714–2727.

Jingyi Zhang, Jiaxing Huang, Sheng Jin, and Shijian Lu. 2024a. Vision-language models for vision tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*.

Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2024b. Sciencebenchmark: A complex real-world benchmark for evaluating natural language to sql systems. *Proceedings of the VLDB Endowment*, 17(4):685–698.

A M^{2}EX on ArtWork


Plot the number of paintings that depict war for each century.
i Planning & Expert Model Allocation
GPT-4o
workflow plan
t_1 text2SQL Retrieve the image paths and the century of paintings from the database. db_schema
t_2 image_analysis Does the image depict war? $\$t_1$
t_3 data_preparation Prepare data to count the number of paintings depicting war for each century. $\$t_2$
t_4 data_ploting Plot the number of paintings depicting war for each century. $\$t_3$
Expert Models & Tools
Data Lake
text2SQL
image_analysis
BLIP
data_preparation
python
data_plotting
Matplotlib
ArtWork DB
ii Execution and Self-Debugging
t_1
t_2
t_3
t_4
paintings
img_path
inception
...
img_01.jpg
image_78.jpg
iii Decision Making
Number of War Paintings by Century
Number of War Paintings
Century
16
18

Reasoning: To retrieve the image paths and determine the century of paintings, we need to extract the 'img_path' and ....

```
SELECT img_path, (CAST(strftime('%Y', inception) AS INTEGER) - 1) / 100 + 1 AS century FROM paintings;
```

```
[{"img_path": "images/img_57.jpg", "century": 16}, .... ,
{"img_path": "images/img_78.jpg", "century": 18,} .... ]
```

```
[{'img_path': 'images/img_0.jpg', 'century': 15, 'Does the image depict war?': 'no'}, ...,
{ "img_path": "images/img_78.jpg", "century": 18, "Does the image depict war?": "yes" }, ...]
```

Reasoning: The data provided includes information about paintings, their centuries, and whether they depict war. To count the number of paintings depicting war for each century, we need to filter the data ...

```
{'War paintings count by century': {16: 1, 18: 2}}
```

Reasoning: The data provided shows the number of war paintings for the 16th and 18th centuries. A bar chart is suitable for visualizing this data as it allows for easy comparison of the number of paintings across different centuries. The plot will be saved to the specified directory.

```python
import matplotlib.pyplot as plt
# Data
centuries = [16, 18]
paintings_count = [1, 2]
#  Plot
plt.figure(figsize=(8, 6))
plt.bar(centuries, paintings_count, color='skyblue')
plt.xlabel('Century')
plt.ylabel('Number of War Paintings')
plt.title('Number of War Paintings by Century')
plt.xticks(centuries)
plt.savefig('plot-out.png')
plt.close()
```

Thought: The data preparation and plotting steps have been successfully completed, and the plot has been saved. I can now provide a summary of the findings.

Summary: "The number of paintings depicting war has been plotted for the 16th and 18th centuries.",
"Details": "The analysis identified 1 painting from the 16th century and 2 paintings from the 18th century that depict War. The plot visualizes these findings.",
"source": "The information was obtained through image analysis and data preparation steps, followed by plotting using matplotlib.",
"extra explanation": "The plot is saved as an image file and can be accessed for visual reference."

Figure 3: M^{2}EX framework on ArtWork (Urban and Binnig, 2024) with an example of processing a multi-modal query. The query is automatically decomposed into various components such as text2SQL, and image analysis which can be inspected by the user for explainability.

B Prompts

Planner Prompt / Replanning Prompt

```
[SYSTEM]: Given a user question and a database schema, analyze the question to identify and break it down into relevant sub-questions.
Determine which tools (e.g., {tool_names}) are appropriate for answering each sub-question based on the available database information and
tools.
Decompose the user question into sub-questions that capture all elements of the question's intent. This includes identifying the main objective,
relevant sub-questions, necessary background information, assumptions, and any secondary requirements.
Ensure that no part of the original question's intent is omitted, and create a list of individual steps to answer the question fully and
accurately using tools.
You may need to use one tool multiple times to answer the original question.
First, you should begin by thoroughly analyzing the user's main question. It's important to understand the key components and objectives within
the query.
Next, you must review the provided database schema. This involves examining the tables, fields, and relationships within the database to
identify which parts of the schema are relevant to the user's question and contribute to a set of sub-questions.
For each sub-question, provide all the required information that may required in other tasks. In order to find this information look at the
user question and the database information.
Each sub-question or step should focus exclusively on a single task.
Each sub-question should be a textual question. Don't generate a code as a sub-question.
Create a plan to solve it with the utmost parallelizability.
Each plan should comprise an action from the following {num_tools} types:
{tool_descriptions}
{num_tools}. join(): Collects and combines results from prior actions.
- An LLM agent is called upon invoking join() to either finalize the user query or wait until the plans are executed.
- join should always be the last action in the plan, and will be called in two scenarios:
(a) if the answer can be determined by gathering the outputs from tasks to generate the final response.
(b) if the answer cannot be determined in the planning phase before you execute the plans. Guidelines:
- Each action described above contains input/output types and descriptions.
- You must strictly adhere to the input and output types for each action.
- The action descriptions contain the guidelines. You MUST strictly follow those guidelines when you use the actions.
- Each action in the plan should strictly be one of the above types. Follow the Python conventions for each action.
- Each action MUST have a unique ID, which is strictly increasing.
- Inputs for actions can either be constants or outputs from preceding actions. In the latter case, use the format $id to denote the ID of the
previous action whose output will be the input.
- If there is an input from preceding actions, always point its id as '$id' in the context of the action
- Always call join as the last action in the plan. Say '<END_OF_PLAN>' after you call join.
- Ensure the plan maximizes parallelizability.
- Only use the provided action types. If a query cannot be addressed using these, invoke the join action for the next steps.
- Never introduce new actions other than the ones provided.
{list of usecase-specific business rules}
[USER]:{state}
[SYSTEM]: Remember, ONLY respond with the task list in the correct format! E.g.: idx. tool(arg_name=args),
```

Prompt for Decision Making

```
[SYSTEM]: Solve a question answering task. Here are some guidelines:
- In the Assistant Scratchpad, you will be given results of a plan you have executed to answer the user's question.
- Thought needs to reason about the question based on the Observations in 1-2 sentences.
- Ignore irrelevant action results.
- If the required information is present, give a concise but complete and helpful answer to the user's question. - If you are unable to give a
satisfactory finishing answer, replan to get the required information. Respond in the following format:
Thought: <reason about the task results and whether you have sufficient information to answer the question>
Action: <action to take>
- If an error occurs during previous actions, replan and take corrective measures to obtain the required information.
- Ensure that you consider errors in all the previous steps, and try to replan accordingly.
- Ensure the final answer is provided in a structured format as JSON as follows:
{{'Summary': <concise summary of the answer>,
'details': <detailed explanation and supporting information>,
'source': <source of the information or how it was obtained>,
'inference':<your final inference as YES, No, or list of requested information without any extra information which you can take from the 'labels'
as given below>, 'extra explanation':<put here the extra information that you don't provide in inference >,
}}
In the 'inference' do not provide additional explanation or description. Put them in 'extra explanation'.
Available actions:
(1) Finish (the final answer to return to the user): returns the answer and finishes the task.
(2) Replan(the reasoning and other information that will help you plan again. Can be a line of any length): instructs why we must replan.
[USER]: {state}
[SYSTEM]: Using the above previous actions, decide whether to replan or finish.
If all the required information is present, you may finish. Consider replanning for data_preparation task if you want to structure the response
in a proper way.
If you have made many attempts to find the information without success, admit so and respond with whatever information you have gathered so the
user can work well with you.
Do not generate a response based on the sample data (assumption). If you failed after multiple attempts, you can finish and explain the reason.
```

Prompt for text2SQL

```
[SYSTEM]: You are a database expert. Generate a SQL query given the following user question, database information and other context that you
receive. You should analyse the question, context and database schema and come up with the executable sqlite3 query.
Provide all the required information in the SQL code to answer the original user question that may required in other tasks utilizing the relevant
database schema.
Ensure you include all necessary information, including columns used for filtering, especially when the task involves plotting or data
exploration.
This must be taken into account when performing any time-based data queries or analyses.
Translate a text question into a SQL query that can be executed on the SQLite database.
You should stick to the available schema including tables and columns in the database and should not bring any new tables or columns.
[USER]: {text2SQL task description}, {db schema}
```

Prompt for text_analysis

```
[SYSTEM]: You are a text analysis assistant. Analyze the provided question and report to answer the question.
Only answer the question and don't provide extra information in your answer.
In your answer, be concrete and use None if you can't find the answer in the report.
The output should be in the format: {{'reasoning': '...', 'answer': '...'}}
[USER]: {text analysis task description}, {text}
```

Prompt for data_preparation

```
[SYSTEM]: You are a data preparation and processing assistant. Create a proper structure for the provided data from the previous steps to answer
the request.
- If the required information has not found in the provided data, ask for replanning and ask from previous tools to include the missing
information.
- You should include all the input data in the code, and prevent of ignoring them by '# ... (rest of the data)'.
- You should provide a name or caption for each value in the final output considering the question and the input context."
- Don't create any sample data in order to answer to the user question.
- You should print the final data structure.
- You should save the final data structure at the specified path with a proper filename.
- You should output the final data structure as a final output.
[USER]: {data preparation task description}, {result from previous task}
```

Prompt for data_plotting

```
[SYSTEM]: You are a data plotting assistant. Plot the provided data from the previous steps to answer the question.
- Analyze the user's request and input data to determine the most suitable type of visualization/plot that also can be understood by the simple
user.
- If the required information has not been found in the provided data, ask for replanning and ask from previous tools to include the missing
information.
- Don't create any sample data in order to answer to the user question.
- You should save the generated plot at the specified path with the proper filename and .png extension.
[USER]: {data plotting task description}, {data}
```

C Tools, Models, and Prompts by Subtask

Task	Tool / Model	Prompt Type
Text-to-SQL translation	GPT-4o	text2SQL prompt
Text analysis	GPT-4o	text_analysis prompt
ArtWork VQA	BLIP-2	no prompt
Medical image (EHRXQA) VQA	M3AE	no prompt
Data preparation	GPT-4o and Python (Pandas)	data_preparation pormpt and Code via LLM output
Plot generation	GPT-4o and Matplotlib + Pandas	data_plotting prompt and Chart Code via LLM output
DAG construction (planning/replannig)	GPT-4o (Planner loop)	planner Prompt / replanning prompt
Decision Making	GPT-4o	decision making prompt

Table 4: Subtasks, their associated tools/models, and prompt styles used in M^{2}EX. Most tool invocations are zero-shot or template-based.

D Optimizations in M^{2}EX Explained with Examples

To better demonstrate advantages of M^{2}EX, we provide several examples (see Figures 3 and 4) across three key aspects: *explanations, smart replanning, and parallel planning*. The following examples provide a detailed illustration of these three aspects.

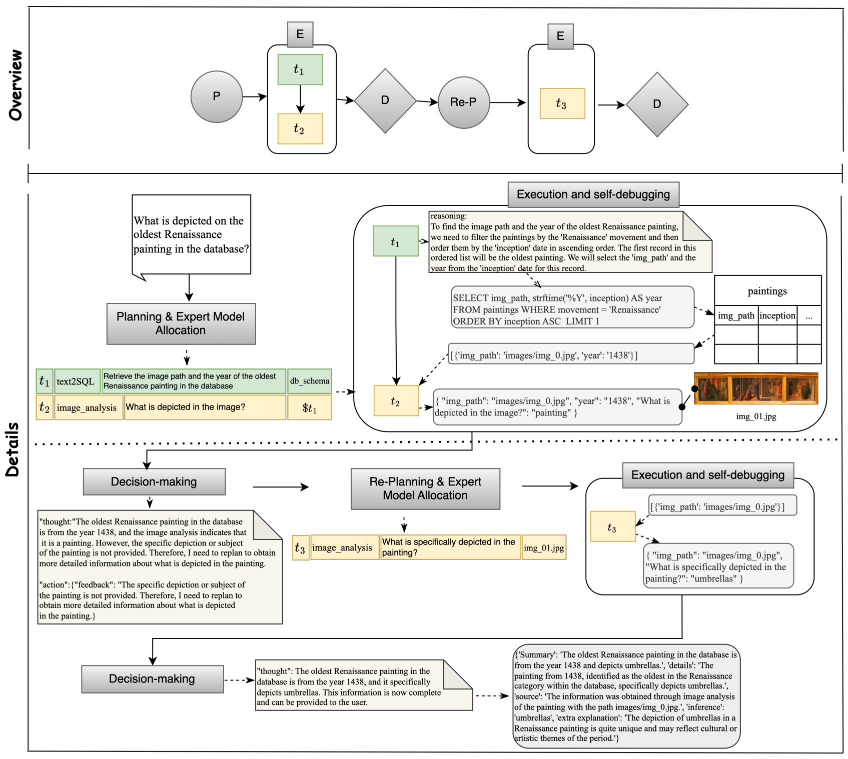
Example 1: *Plot the number of paintings that depict war for each century* (see Figure 3).

Through a series of well-planned and systematically executed steps, the model demonstrates not only how it processes the query but also how it provides transparency and reasoning at every stage, ensuring the user understands the process and results. The figure depicts a workflow that involves (1) Planning & Expert Model Allocation, (2) Execution & Self-Debugging, and (3) Decision Making. Here's a breakdown of each step:
1) Planning & Expert Model Allocation: The process begins with the query being broken down into a sequence of subtasks: Task 1: Retrieve painting metadata, including their years and associated centuries, from the database. Task 2: Analyze the images to determine whether they depict war. Task 3: Prepare the data by counting the number of war-related paintings per century. Task 4: Visualize these counts in a bar chart.

Each task is allocated to specialized tools or models, such as text2SQL to translate the natural language question to SQL and database retrieval, image analysis tools for visual interpretation, coding tools to structure the data, and visualization libraries like matplotlib. This stage establishes a clear plan, showing how the overall query will be tackled in logical steps.
2) Execution & Self-Debugging: The model begins executing the tasks, providing explanations and outputs at every stage to ensure clarity. Task 1 - Retrieving Data: The model constructs a SQL query to retrieve the required information from the database. It explains its reasoning: to determine the century of each painting, it converts the inception year into century values. The result is a list of paintings, each associated with its image path and century. Task 2 - Image Analysis: With the retrieved data, the model analyzes each painting to determine if it depicts war. It applies image analysis tools to interpret the visual content of the paintings. The reasoning here is clear—war-related imagery, such as battles or soldiers, must be identified to answer the query. The output is a dataset indicating whether each painting depicts war. Task 3 - Data Preparation: The model filters and aggregates the data, counting the number of paintings depicting war for each century. It explains that grouping the paintings by century allows for easy comparison of trends across time periods. The result is a concise summary: `1 painting from the 16th century and 2 from the 18th century are identified as depicting war.` Task 4 - Data Visualization: Finally, the model prepares a bar chart to visualize the results. It explains its reasoning for choosing this visualization: bar charts effectively compare counts across categories, in this case, centuries. A Python script is provided, showing how the chart was generated, and the output is saved as an image for user reference.
3) Decision Making: When the tasks are completed, the model reflects on its work and provides a final

output based on its thought as `Summary:"The number of paintings depicting war has been plotted for the 16th and 18th centuries.", "Details": "The analysis identified 1 painting from the 16th century and 2 paintings from the 18th century that depict war. The plot visualizes these findings. [..]"`. Throughout the workflow, the model demonstrates a commitment to transparency.

At every stage, M^{2}EX provides reasoning to justify its actions, from choosing SQL for retrieval to selecting a bar chart for visualization. Intermediate outputs, like the dataset of war paintings and the Python plotting code, are made visible, ensuring the user can trace the steps taken. The decision making phase wraps up the process by summarizing findings, clarifying the approach, and sharing the final visual result. This shows that M^{2}EX not only answers the query effectively but also ensures its steps are understandable, logical, and well-documented, building trust in its analysis.


Overview
P
E
t_1
t_2
D
Re-P
E
t_3
D
Details
Planning & Expert Model Allocation
Execution and self-debugging
paintings
img_path
inception
...
img_01.jpg
Decision-making
Re-Planning & Expert Model Allocation
Execution and self-debugging
Decision-making

What is depicted on the oldest Renaissance painting in the database?

t_1	text2SQL	Retrieve the image path and the year of the oldest Renaissance painting in the database	db_schema
t_2	image_analysis	What is depicted in the image?	$\$t_1$

reasoning:
To find the image path and the year of the oldest Renaissance painting, we need to filter the paintings by the 'Renaissance' movement and then order them by the 'inception' date in ascending order. The first record in this ordered list will be the oldest painting. We will select the 'img_path' and the year from the 'inception' date for this record.

```
SELECT img_path, strftime('%Y', inception) AS year
FROM paintings WHERE movement = 'Renaissance'
ORDER BY inception ASC  LIMIT 1
```

```
[{'img_path': 'images/img_0.jpg', 'year': '1438'}]
```

```
{ "img_path": "images/img_0.jpg", "year": "1438", "What is depicted in the image?": "painting" }
```

"thought:"The oldest Renaissance painting in the database is from the year 1438, and the image analysis indicates that it is a painting. However, the specific depiction or subject of the painting is not provided. Therefore, I need to replan to obtain more detailed information about what is depicted in the painting.

"action":{"feedback": "The specific depiction or subject of the painting is not provided. Therefore, I need to replan to obtain more detailed information about what is depicted in the painting.}

t_3	image_analysis	What is specifically depicted in the painting?	img_01.jpg

```
[{'img_path': 'images/img_0.jpg'}]
```

```
{ "img_path": "images/img_0.jpg", "What is specifically depicted in the painting?": "umbrellas" }
```

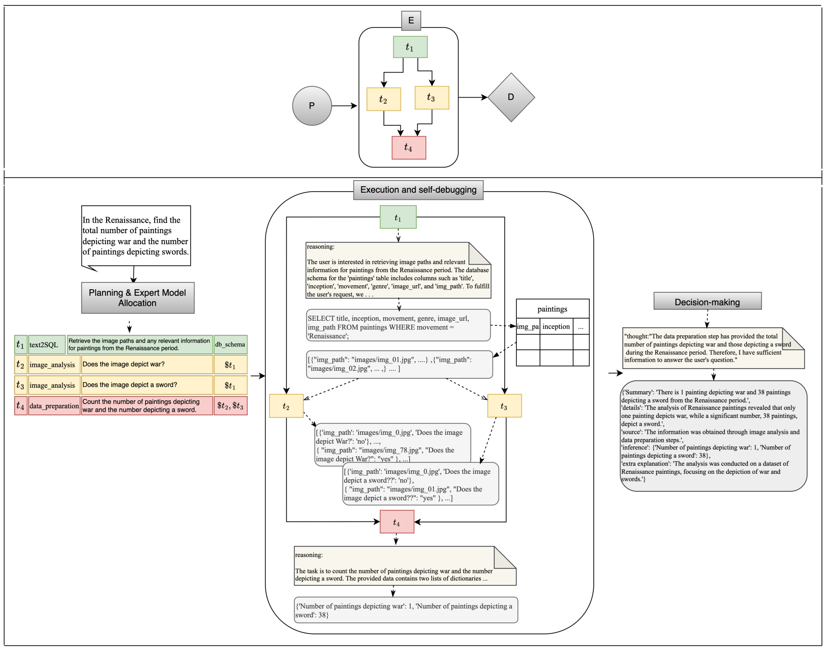
"thought": The oldest Renaissance painting in the database is from the year 1438, and it specifically depicts umbrellas. This information is now complete and can be provided to the user.

{'Summary': 'The oldest Renaissance painting in the database is from the year 1438 and depicts umbrellas.', 'details': 'The painting from 1438, identified as the oldest in the Renaissance category within the database, specifically depicts umbrellas.', 'source': 'The information was obtained through image analysis of the painting with the path images/img_0.jpg.', 'inference': 'umbrellas', 'extra explanation': 'The depiction of umbrellas in a Renaissance painting is quite unique and may reflect cultural or artistic themes of the period.'}

Figure 4: Optimization of M^{2}EX: Smart replanning.

Example 2 - Smart Replanning: *What is depicted on the oldest Renaissance painting in the database?* (see Figure 4).

Contrary to the previous example, M^{2}EX here involves *smart replanning* - a major optimization technique of M^{2}EX. The main idea is to dynamically adapt the planning in case some tasks of the workflow fail or do not produce any results. Here's a breakdown of each step:
1) Planning & Expert Model Allocation: M^{2}EX outputs the initial workflow plan that has 2 tasks. The first task involves retrieving the image path and the year of the oldest Renaissance painting in the database using a "text2SQL" expert model. It also involves an "image_analysis" expert model in the second task, which aims to determine what is depicted in the image.
2) Execution and Self-Debugging: M^{2}EX takes the information about the planned workflow as well as task dependencies and puts it into action. In Task 1, it comes with a reasoning statement to generate the SQL query as: `SELECT img_path, strftime('%Y', inception) AS year FROM paintings WHERE movement = 'Renaissance' ORDER BY inception ASC LIMIT 1`. Then it executes the query over the Artwork database and retrieves the specific image path and year for the oldest Renaissance painting as `['img_path': 'images/img_0.jpg', 'year': '1438']`. This allows the model to access the actual painting data in the subsequent task.



In the Renaissance, find the total number of paintings depicting war and the number of paintings depicting swords.

t_1	text2SQL	Retrieve the image paths and any relevant information for paintings from the Renaissance period.	db_schema
t_2	image_analysis	Does the image depict war?	$\$t_1$
t_3	image_analysis	Does the image depict a sword?	$\$t_1$
t_4	data_preparation	Count the number of paintings depicting war and the number depicting a sword.	$\$t_2, \$t_3$

reasoning:

The user is interested in retrieving image paths and relevant information for paintings from the Renaissance period. The database schema for the 'paintings' table includes columns such as 'title', 'inception', 'movement', 'genre', 'image_url', and 'img_path'. To fulfill the user's request, we . . .

```
SELECT title, inception, movement, genre, image_url,
img_path FROM paintings WHERE movement =
'Renaissance';
```

```
[{"img_path": "images/img_01.jpg", ....} ,{"img_path":
"images/img_02.jpg", ... ,} .... ]
```

```
[{'img_path': 'images/img_0.jpg', 'Does the image
depict War?': 'no'}, ...,
{ "img_path": "images/img_78.jpg", "Does the
image depict War?": "yes" }, ...]
```

```
[{'img_path': 'images/img_0.jpg', 'Does the image
depict a sword??': 'no'},
{ "img_path": "images/img_01.jpg", "Does the
image depict a sword??": "yes" }, ...]
```

reasoning:

The task is to count the number of paintings depicting war and the number depicting a sword. The provided data contains two lists of dictionaries ...

```
{'Number of paintings depicting war': 1, 'Number of paintings depicting a
sword': 38}
```

"thought:"The data preparation step has provided the total number of paintings depicting war and those depicting a sword during the Renaissance period. Therefore, I have sufficient information to answer the user's question."

```
{'Summary': 'There is 1 painting depicting war and 38 paintings
depicting a sword from the Renaissance period.',
'details': 'The analysis of Renaissance paintings revealed that only
one painting depicts war, while a significant number, 38 paintings,
depict a sword.',
'source': 'The information was obtained through image analysis and
data preparation steps.',
'inference': {'Number of paintings depicting war': 1, 'Number of
paintings depicting a sword': 38},
'extra explanation': 'The analysis was conducted on a dataset of
Renaissance paintings, focusing on the depiction of war and
swords.'}
```

Figure 5: Optimization of M^{2}EX: Parallel planning.

In Task 2, M^{2}EX utilizes the "image_analysis" expert model (i.e. visual question answering based on BLIP) to examine the contents of `img_0.jpg` to answer the question: *What is depicted in the image?* The output of this task is transferred as a final result to the decision making component. At this point, the model's "thought" process in this component becomes evident. It reasons that while it knows that `img_0.jpg` is a painting, the details about what is depicted in the painting have not been provided. Therefore, the model decides to not provide a final answer to the user and does replanning.

The replanning capability is a crucial aspect of the M^{2}EX's approach. Rather than blindly accepting the final answer which does not produce a satisfiable or correct result, the model recognizes the need to replan and calls the "image_analysis" module again. Since the model already knows which image in the database contains the oldest Renaissance painting, it smartly plans the "image_analysis" task as Task 3, by reformulating the question as *What is specifically depicted in the painting?* M^{2}EX then executes the task, and receives the more concrete answer "umbrellas".

Moving forward, the decision making component confirms the details about the painting. Here, it verifies that the information it has gathered so far aligns with the natural language question and makes sense as a comprehensive understanding of the oldest Renaissance painting. The key aspect is the model's ability to replan effectively and to strategically leverage the available information to avoid repeating tasks.

Example 3 - Parallel Planning: *In the Renaissance, find the total number of paintings depicting war and the number of paintings depicting swords* (see Figure 5).

The figure illustrates how M^{2}EX processes a complex query about Renaissance paintings, focusing on identifying how many paintings depict war and how many depict swords. The pipeline is structured to combine *parallel task execution with step-by-step explanations*, ensuring clarity and efficiency throughout the process.

The process begins in the Planning & Expert Model Allocation, where the model breaks down the user's query into distinct subtasks. These subtasks are assigned to specialized modules: Task 1 "text2SQL": This task retrieves image paths and relevant metadata for Renaissance paintings from a database using a SQL query. Task 2 "image_analysis": This task examines whether each painting depicts war. Task 3 "image_analysis": Simultaneously, another module analyzes whether each painting depicts a sword. Task 4 "data_preparation": This task consolidates the results from Task 2 and Task 3 to count and summarize the paintings.

The execution phase begins with Task 1, where the model generates and runs a SQL query. The reasoning provided for this step explains how the schema is understood and how the query ensures that only Renaissance paintings are retrieved. The output of Task 1 includes image paths and metadata, which are then sent to the next stage.

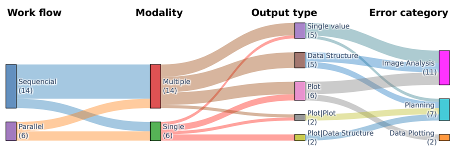
At this point, the model showcases its parallel planning capability. Tasks 2 and 3 are performed concurrently: For Task 2, the system uses image analysis to determine if each painting depicts war. For Task 3, a similar image analysis process identifies paintings that depict swords. Running these tasks in parallel significantly speeds up the workflow, as they operate independently of each other. Once the image analysis tasks are complete, the model transitions to Task 4, where it aggregates the results. The reasoning here details how the system compiles two lists - one for paintings depicting war and one for those depicting swords. Afterwards, M^{2}EX counts the entries in each list. The final results are prepared for the decision making module.

In the decision making phase, the model reflects on its findings. It confirms that sufficient data was processed to answer the query and provides a summary: `"There is 1 painting depicting war and 38 paintings depicting swords."`

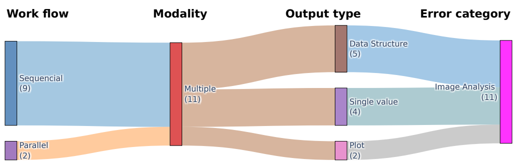
M^{2}EX offers details, explaining how the analysis was conducted and highlighting the disparity between the two categories of paintings. The system further provides an explanation of its methodology, emphasizing how it worked systematically to answer the query. This demonstrates M^{2}EX's ability to manage tasks efficiently through parallel execution and to ensure transparency through reasoned explanations at every step. By combining these capabilities, the system provides a clear, accurate, and well-supported response to the user's query.

Note that we did not compare M^{2}EX with NeuralSQL on ArtWork dataset, as such a comparison would be unfair due to NeuralSQL's inability to support plotting.

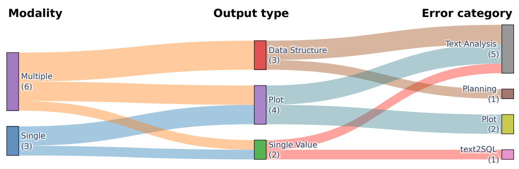
E Error Analysis



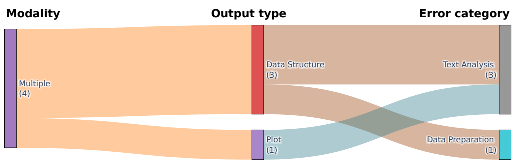
(a) CAESURA (ArtWork)



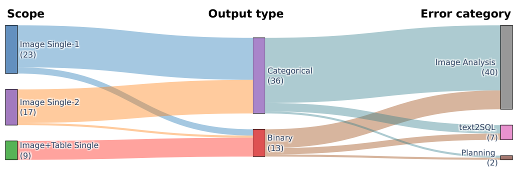
(b) M^{2}EX (ArtWork)



(c) CAESURA (RotoWire)



(d) M^{2}EX (RotoWire)



(e) M^{2}EX (EHRXQA)

Figure 6: Error analysis on different datasets: (a) CAESURA on ArtWork, (b) M^{2}EX on ArtWork, (c) CAESURA on RotoWire, (d) M^{2}EX on RotoWire, and (e) M^{2}EX on EHRXQA.

Error Analysis on the ArtWork Dataset As illustrated in Figure 6 (a), a total of 20 errors are identified out of 30 inference tasks for CAESURA. Of these, 14 errors occur within CAESURA's sequential workflow. The errors include three single-modal questions and 11 multi-modal questions. Among the three single-modal, one task could not be resolved due to insufficient data available in the data pool. Following this failure, CAESURA attempts to replan twice but ultimately generates an incorrect plan, and consequently results in an erroneous response. The remaining two errors in single-modal tasks were classified as *Plot Generation Errors*, which are caused by inconsistencies in the time axis units of the plot output.

For 11 errors in multi-modal questions, five are related to single-value outputs, four to plots, and three to data structures. All of these errors are attributed to incorrect outputs generated by the image analysis model. After further research, we found two ambiguous tasks in classifying the error categories. *(1) Plot the number of paintings that depict war for each year* and *(2) What is depicted on the oldest religious artwork in the database?* Both tasks failed due to improperly parsed sub question for the image analysis task, specifically the oversimplified term "war." While this term is semantically related to the correct natural language question, "Does the image depict war?", it does not fully capture the intent of the task. As a result, it cannot be classified as a completely faulty question. Notably, the M^{2}EX model generated correct results for these tasks, underscoring the limitations of CAESURA's approach in handling subtle semantic distinctions.

In questions which require a parallel workflow - including two data structures, plot | plot, and plot | data structure outputs — errors are observed at the early planning stage. Our analysis reveals that CAESURA encounters significant challenges in generating accurate plans for embarrassingly parallel tasks. For two of these tasks, the system fails to generate any plan at all. For the remaining four tasks, CAESURA can provide partial results for some subtasks, but other subtasks are left unanswered, reflecting a broader issue in its ability to manage parallel planning. Our M^{2}EX system successfully generates the appropriate plans for all tasks, as shown in Figure 6 (b). In addition, all text-to-SQL steps, data preparation pipelines, and plot outputs, where required, are validated as correct. As illustrated in Figure 6(b), the only source of errors is the inaccurate output of the image analysis model, which accounted for 11 errors. No other errors are located in the text-to-SQL task, plot generation, or task planning deficiencies. This analysis highlights the image analysis model as the bottleneck in system performance, underscoring the need for further refinement in its predictive accuracy.

Error Analysis on the RotoWire Dataset Figure 6 (c) reveals that CAESURA encounters 9 errors across 12 inference tasks on the RotoWire dataset. These tasks are evenly divided between single-modal and multi-modal categories. Among the three single-modal tasks, one stumbles due to an SQL query missing essential filter clauses, resulting in inaccurate structured data. The other two, focused on plotting, fail to generate visualizations consistent with the analytical findings.

In the multi-modal group, six tasks face challenges. A task requiring a single-value output is derailed by suboptimal text analysis. Additionally, the Bart model's limited text comprehension hampers two tasks expecting data structure outputs and two others involving plots, all undermined by faulty text interpretation. Another task, aimed at producing a structured output, falters during the planning stage because the strategy cannot be refined within the permitted attempts.

In contrast, our M^{2}EX system, as illustrated in Figure 6 (d), excels by devising suitable plans for all tasks and accurately resolving every single-modal task. However, it encounters issues in four multi-modal tasks: two demanding data structures and one plotting task succumb to flawed text analysis, while a fourth task needing a structured output fails during post-data preparation. Beyond these, no errors arise in text-to-SQL conversions or plot generation. This comparison underscores M^{2}EX's greater resilience while highlighting text analysis as a shared weakness. CAESURA, however, suffers from additional pipeline limitations.

Error Analysis on the EHRXQA Dataset Since NeuralSQL is a one-step approach lacking task planning and explainability, we are unable to localize the source of errors as systematically as in the M^{2}EX or CAESURA systems. Consequently, we focus our error analysis solely on the M^{2}EX system using the EHRXQA dataset.

Figure 6 (e) presents the distribution of 49 errors across various steps, categorized by their respective scopes: *Image Single-1* (23 errors), *Image Single-2* (17 errors), and *Image+Table Single* (9 errors). Among these, 36 errors are associated with the categorical scope, with 20 attributed to *Image Single-1* and 16 to *Image Single-2*. In contrast, errors linked to the binary output type are primarily found in the *Image+Table Single* scope. Specifically, *Image Single-1* contributes three binary errors, *Image Single-2* accounts for one, and *Image+Table Single* includes nine, summing up to 13 binary errors out of the total 49. Considering the uneven distribution of errors across various output types and scopes, we identified inaccurate image analysis — primarily driven by the M3AE model (Chen et al., 2022) — as the main source of errors. Our analysis reveals that errors linked to categorical output types (36) are nearly three times higher than those associated with binary output types (13). This suggests that the error pattern is less related to the task difficulty across different scopes and more influenced by the output type, as binary questions demonstrate a statistically higher success rate compared to categorical ones. Notably, the *Image + Table Single* scope exclusively utilizes binary output types.

To gain a deeper understanding, a step-by-step error analysis reveals that out of the 23 errors in the *Image Single-1* scope, 22 are due to inaccuracies in image analysis, while only one is related to a misstep in the text-to-SQL process. The specific question text for this case is: *"Catalog all the anatomical findings seen in the image, given the first study of patient 11801290 on the first hospital visit."* The generated SQL query fails to include the condition specifying the *first study*, resulting in an incorrect output. In the *Image Single-2* category, 16 out of 17 total errors are due to inaccurate image analysis, with one error attributed to the text-to-SQL step. The specific query in question is: *"Does the second-to-last study of patient 16345504 this year reveal still-present fluid overload/heart failure in the right lung compared to the first study this year?"*. The text-to-SQL task fails to correctly retrieve the *first and last study of this year* as required, instead erroneously returning multiple studies from the current year. In the *Image+Table Single* scope, all nine errors involve binary output types. Of these, six result from inaccurate image analysis, one from incomplete planning, and two from an incorrect text-to-SQL step. The error caused by incomplete planning occurs with the question: *"Did patient 19055351 undergo the combined right and left heart cardiac catheterization procedure within the same month after a chest x-ray revealed any anatomical findings until 2104?"*. In this case, the plan omits the necessary image analysis step, leading to an incorrect final output. During the reasoning stage, instances were identified where an empty output produced a *no* response that coincidentally aligned with the ground truth. However, M^{2}EX's explainability highlights this as a misclassification, as the absence of output was not due to correct reasoning.

Two errors in the *Image+Table Single* category are attributed to text-to-SQL misbehavior. The specific questions causing these errors are: *"Was patient 12724975 diagnosed with hypoxemia until 1 year ago, and did a chest x-ray reveal any tubes/lines in the abdomen during the same period?"* and *"Was patient 10762986 diagnosed with a personal history of tobacco use within the same month after a chest x-ray showing any abnormalities in the aortic arch until 1 year ago?"* In both cases, the SQL queries fail to correctly apply the condition *(since current time) until 1 year ago*, instead treating *1 year ago* as a fixed point in time.

These findings highlight the pivotal role of accurate image analysis in multi-modal data exploration systems. Particularly, they emphasize a formidable challenge associated with categorical outputs. Moreover, the findings underscore the necessity of robust planning and effective SQL query generation to achieve optimal system performance. Addressing these challenges requires advancements in visual reasoning, temporal logic comprehension, and SQL generation, all of which are essential for mitigating errors and enhancing system accuracy.