Automatic LLM Benchmark Analysis for Text2SQL

Authors: Cristian DEGNI, Francesco GIANNUZZO
Supervised by: Prof. Paolo PAPOTTI, Simone PAPICCHIO (PhD)

Project Background & Motivation

🧠 **LLMs** and approaches like **Text-to-SQL** and **Text-to-Code** are transforming how we query databases, especially in complex, enterprise settings.

📊 Existing **public benchmarks** often **fail** to **reflect** the **diversity and complexity** of **real-world datasets**, limiting accurate model evaluation.

It is therefore important to:

🔍 **Evaluate** the **robustness** and **generalization** of models on SQL-centric tasks.

⚖️ Compare **Text-to-Code** systematically **against** established methods like **Text-to-SQL**.

Project Objectives

1) **Develop** a **web application using GRADIO** to make **QATCH** more **user-friendly** and **accessible** for **end users**.

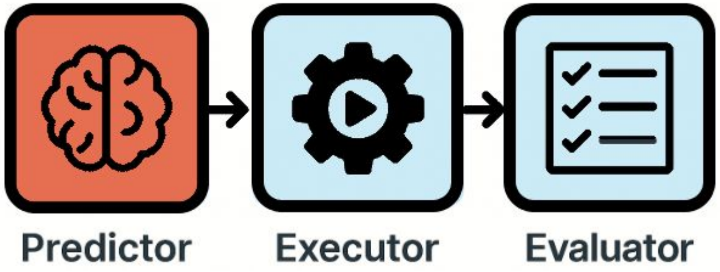
2) **Compare** the **Text-to-SQL and Text-to-Code** approaches using **different LMMs** on **different databases** using **Natural Language Question** for querying the data.

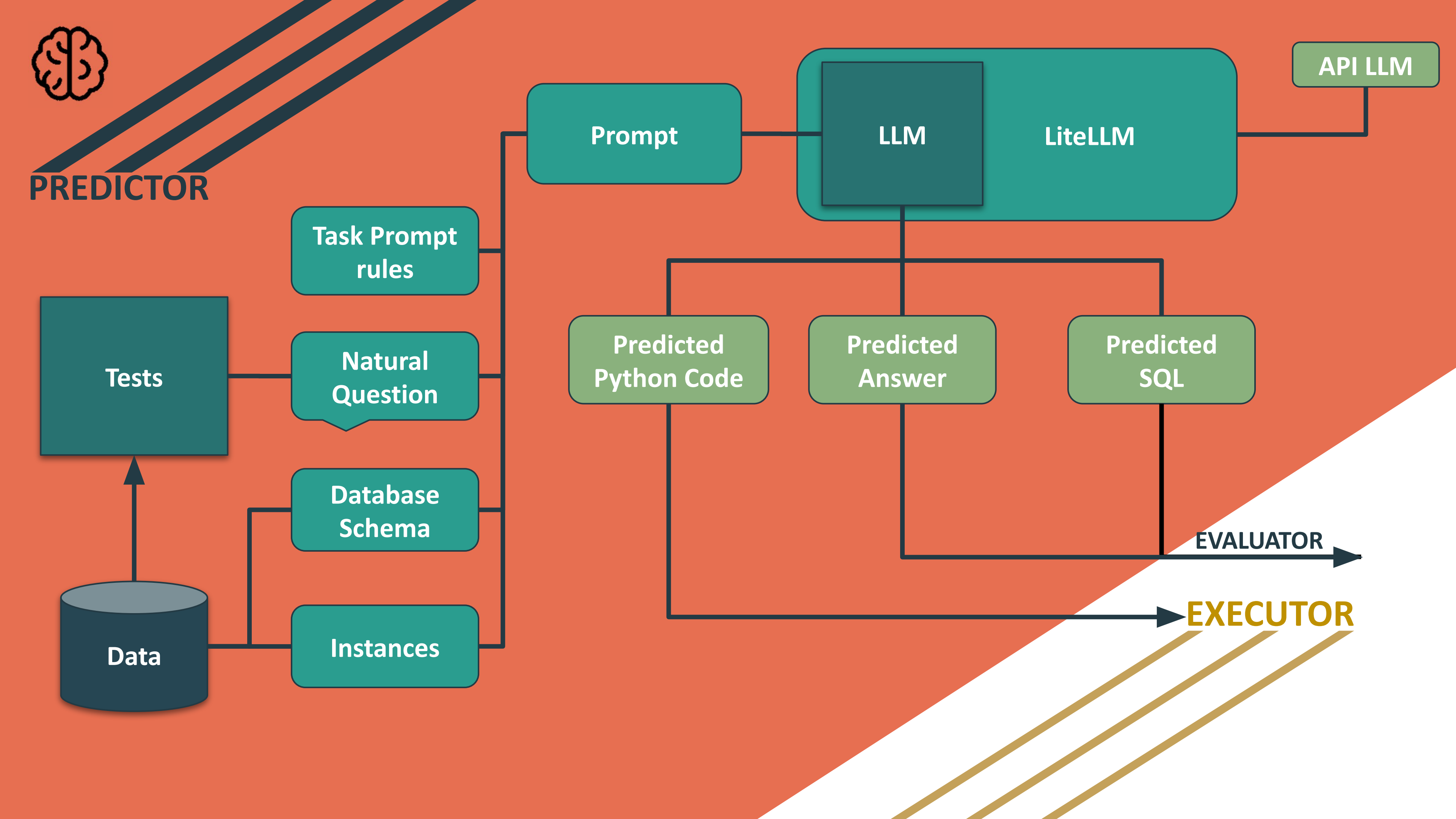
3) **Evaluate** models **performance** using **QATCH metrics** for **multiple scenarios** and **Code metrics** for the **Text-To-Code task.**

4) **Analyze** the **results** obtained from models on **different SQL-centric and non-SQL datasets** to **compare** the **effectiveness** of both task approaches **under realistic conditions**.

Workflow Overview

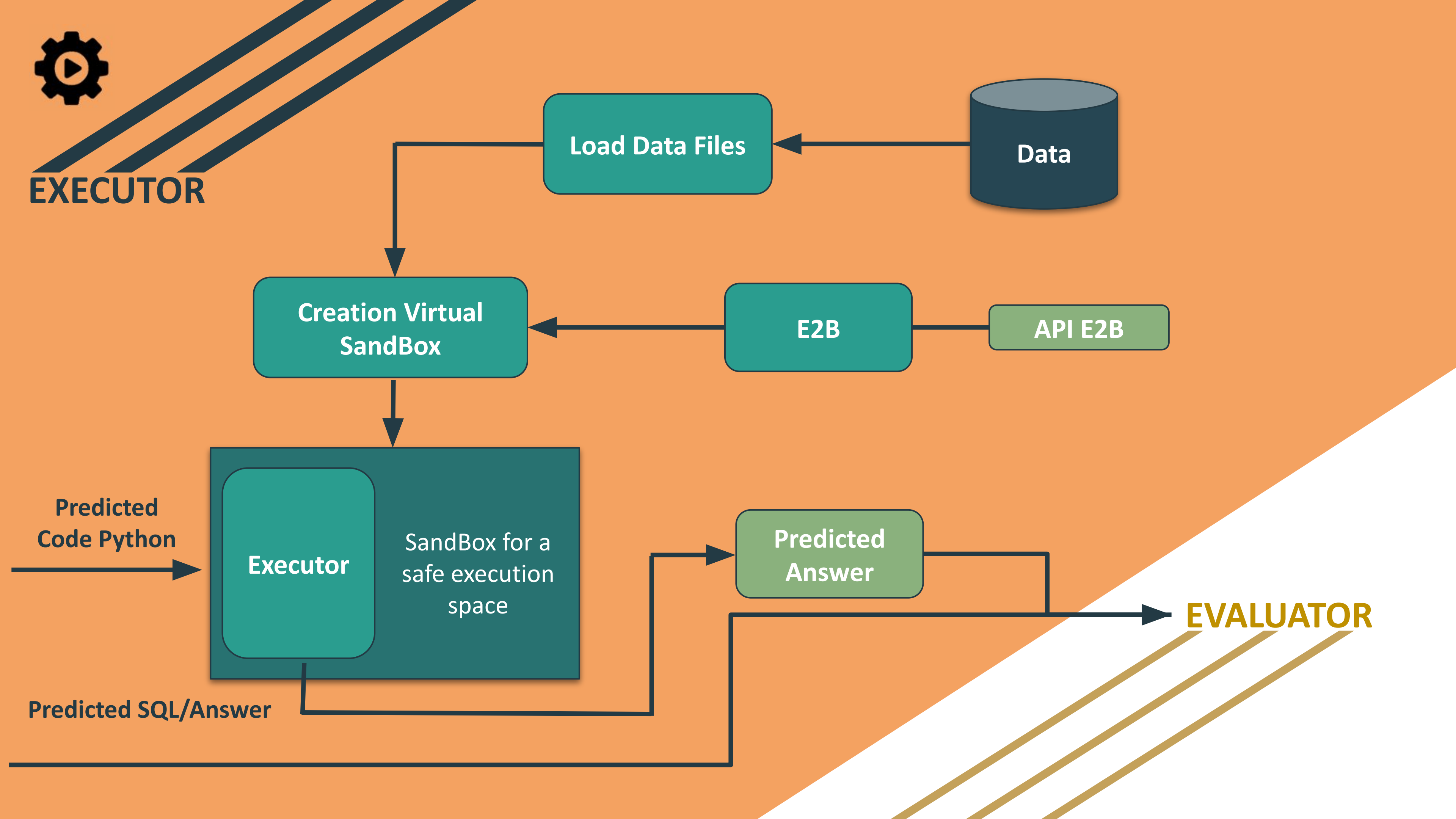
It generates **SQL queries, Python scripts** or **answer**



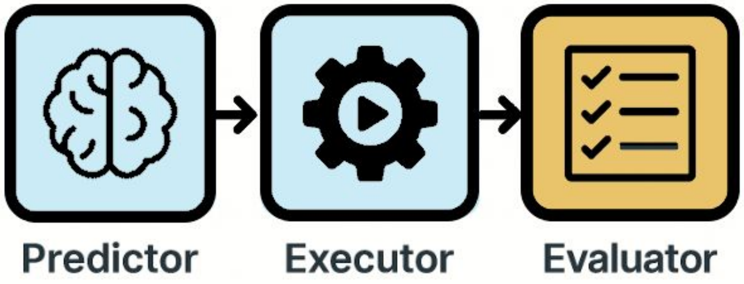

PREDICTOR
Prompt
LLM
LiteLLM
API LLM
Task Prompt rules
Tests
Natural Question
Database Schema
Instances
Data
Predicted Python Code
Predicted Answer
Predicted SQL
EVALUATOR
EXECUTOR

Workflow Overview

It runs those **queries** or **scripts** to produce **result tables**.

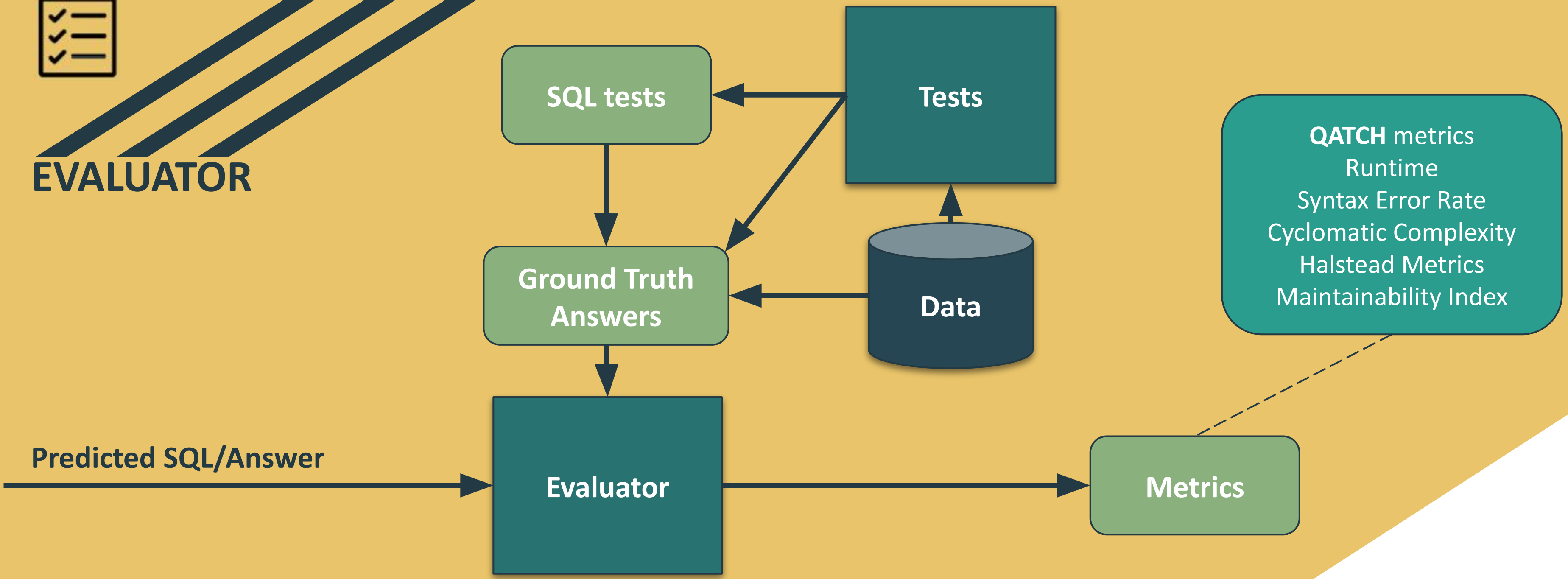

EXECUTOR
Load Data Files
Data
Creation Virtual SandBox
E2B
API E2B
Predicted Code Python
Executor
SandBox for a safe execution space
Predicted Answer
EVALUATOR
Predicted SQL/Answer

Workflow Overview



It compares the **ground truth** tables with predictions for computes **performance metrics**.

EVALUATOR


SQL tests
Tests
QATCH metrics
Runtime
Syntax Error Rate
Cyclomatic Complexity
Halstead Metrics
Maintainability Index
Ground Truth Answers
Data
Predicted SQL/Answer
Evaluator
Metrics

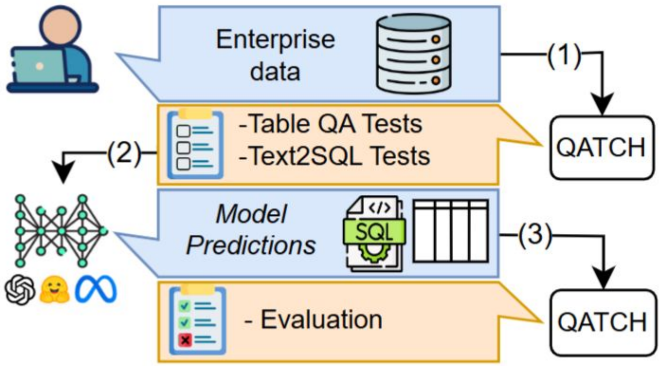
SQL-centric (Text2SQL & Table QA)

Text-to-SQL:

Goal : translate an **NL question** and **data context into** a **SQL query**, which is **executed** to **obtain table results**. **QATCH evaluates** execution accuracy and fine-grained metrics for a full performance profile.

Table Question Answering:

LLM takes an **NL question** and **data context** to produce **directly** an **answer**, without issuing SQL engine. **QATCH** then **compares** this **output to** the **ground-truth answer** to **measure accuracy** and **pinpoint errors** at the cell and tuple levels.



Text2SQL

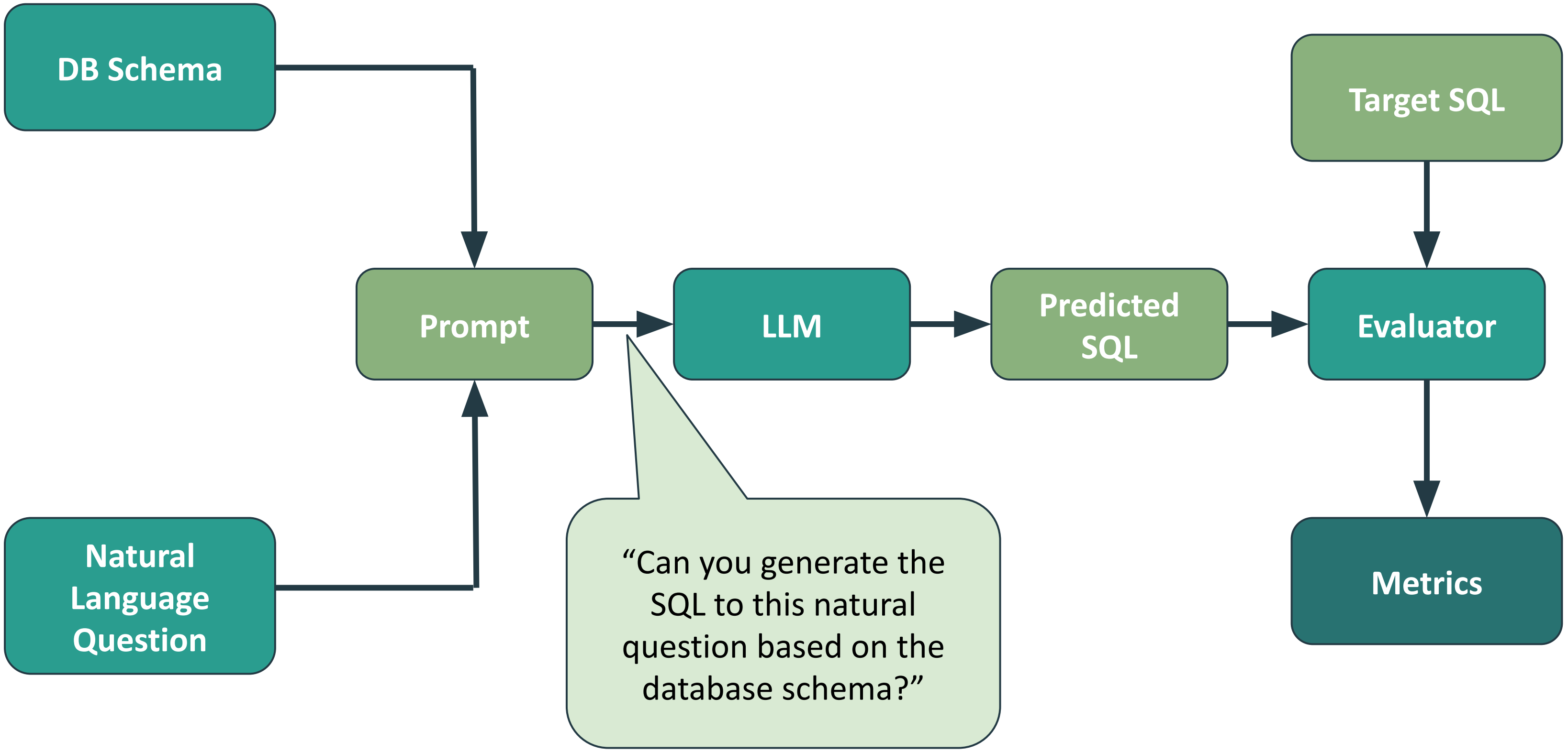
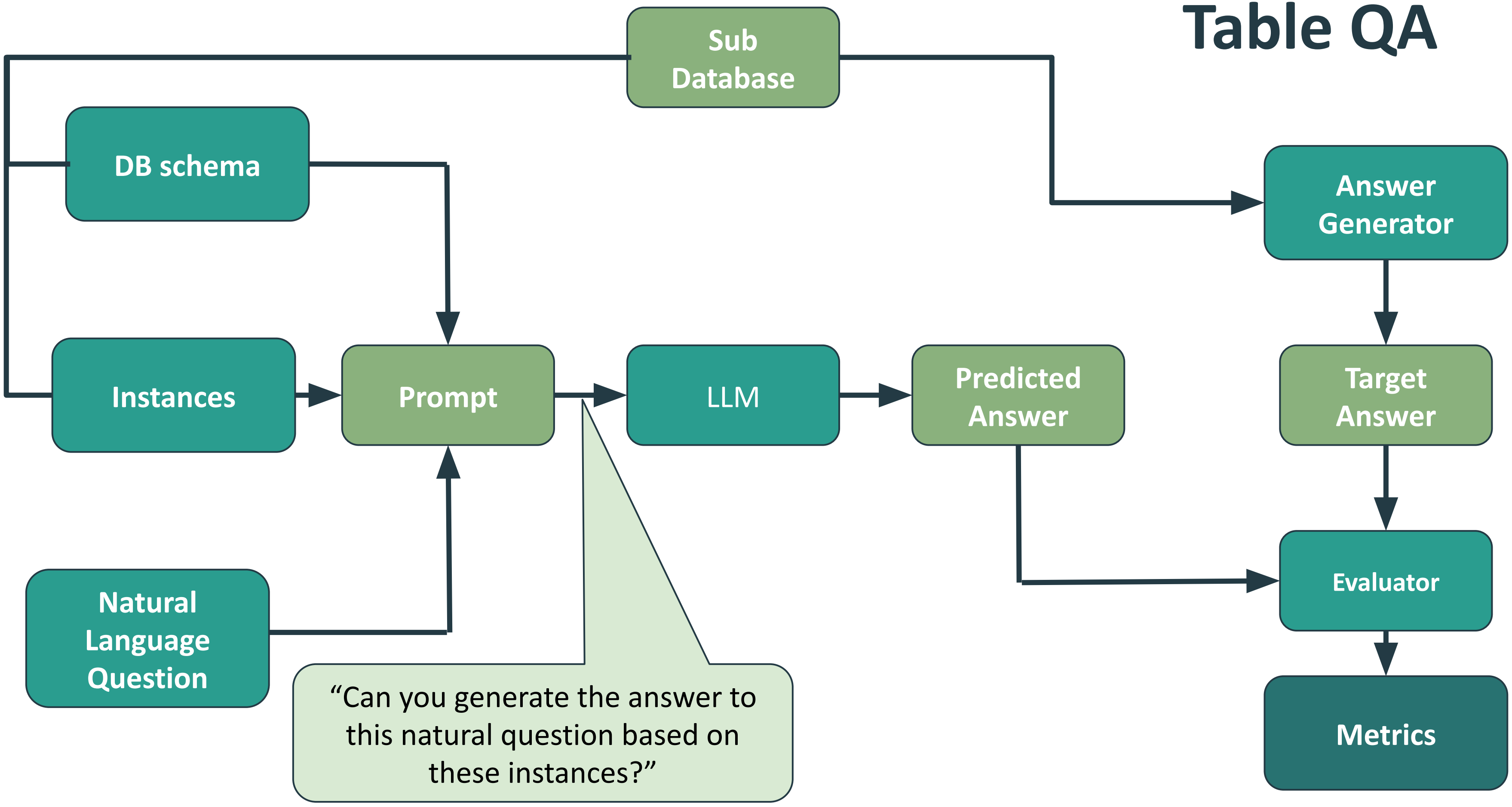

DB Schema
Natural Language Question
Prompt
LLM
Predicted SQL
Target SQL
Evaluator
Metrics
“Can you generate the SQL to this natural question based on the database schema?”

Table QA


Sub Database
DB schema
Answer Generator
Instances
Prompt
LLM
Predicted Answer
Target Answer
Natural Language Question
Evaluator
"Can you generate the answer to this natural question based on these instances?"
Metrics

Text2Code

Definition:

Map a natural language question and database context into executable Python code that returns the answer table.

Approaches:

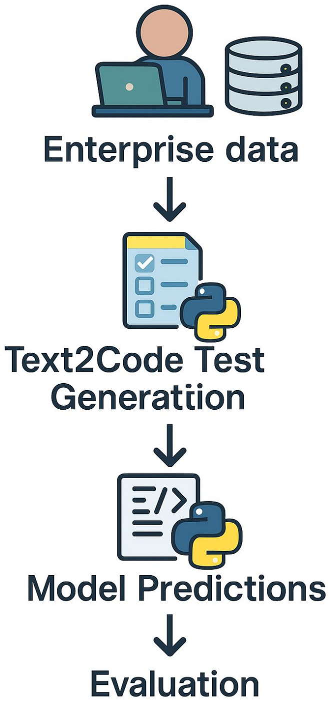
Vanilla: single-prompt for directly generate a Python code

CodexDB: LLM-driven pseudocode translated into Python without leaking gold SQL

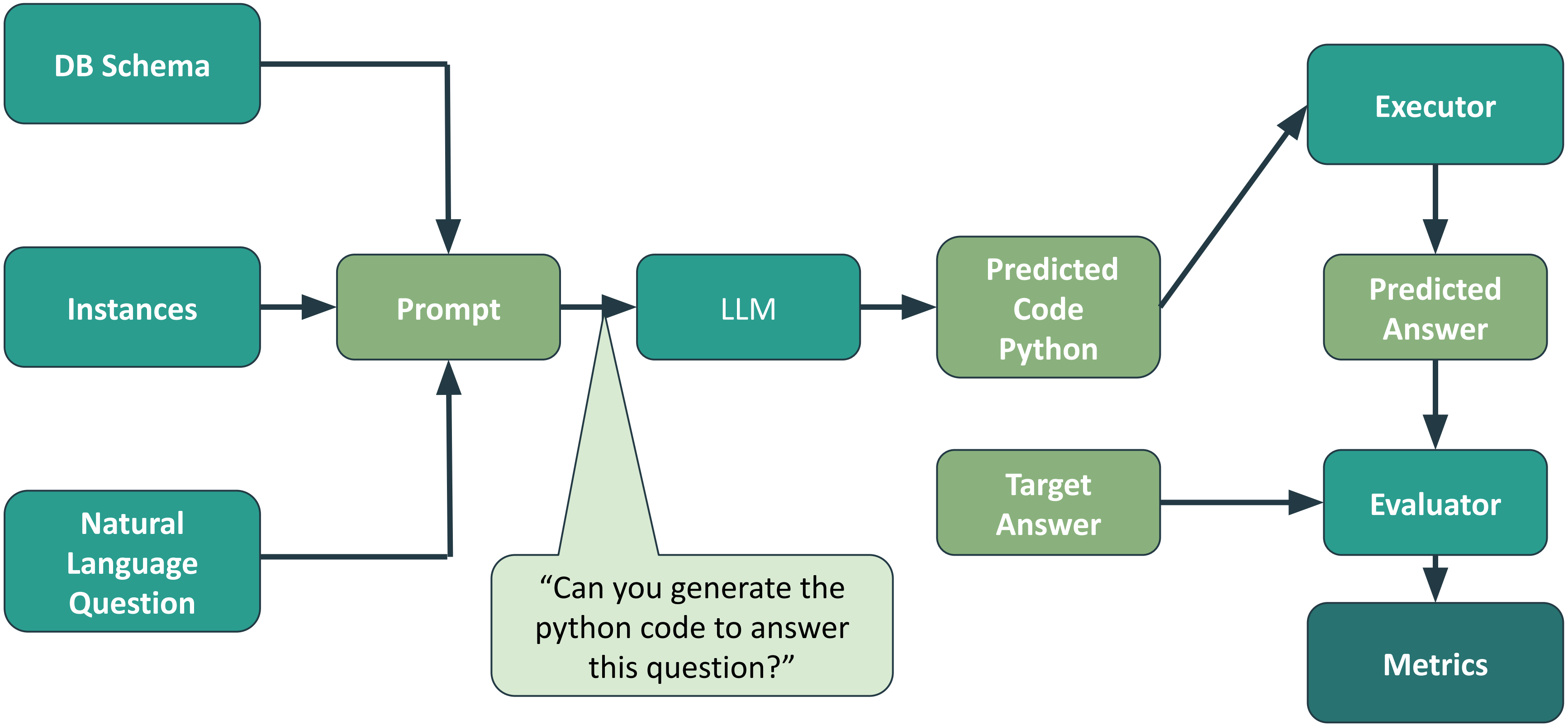
Binder: two-stage LLM decomposition with intermediate mapping

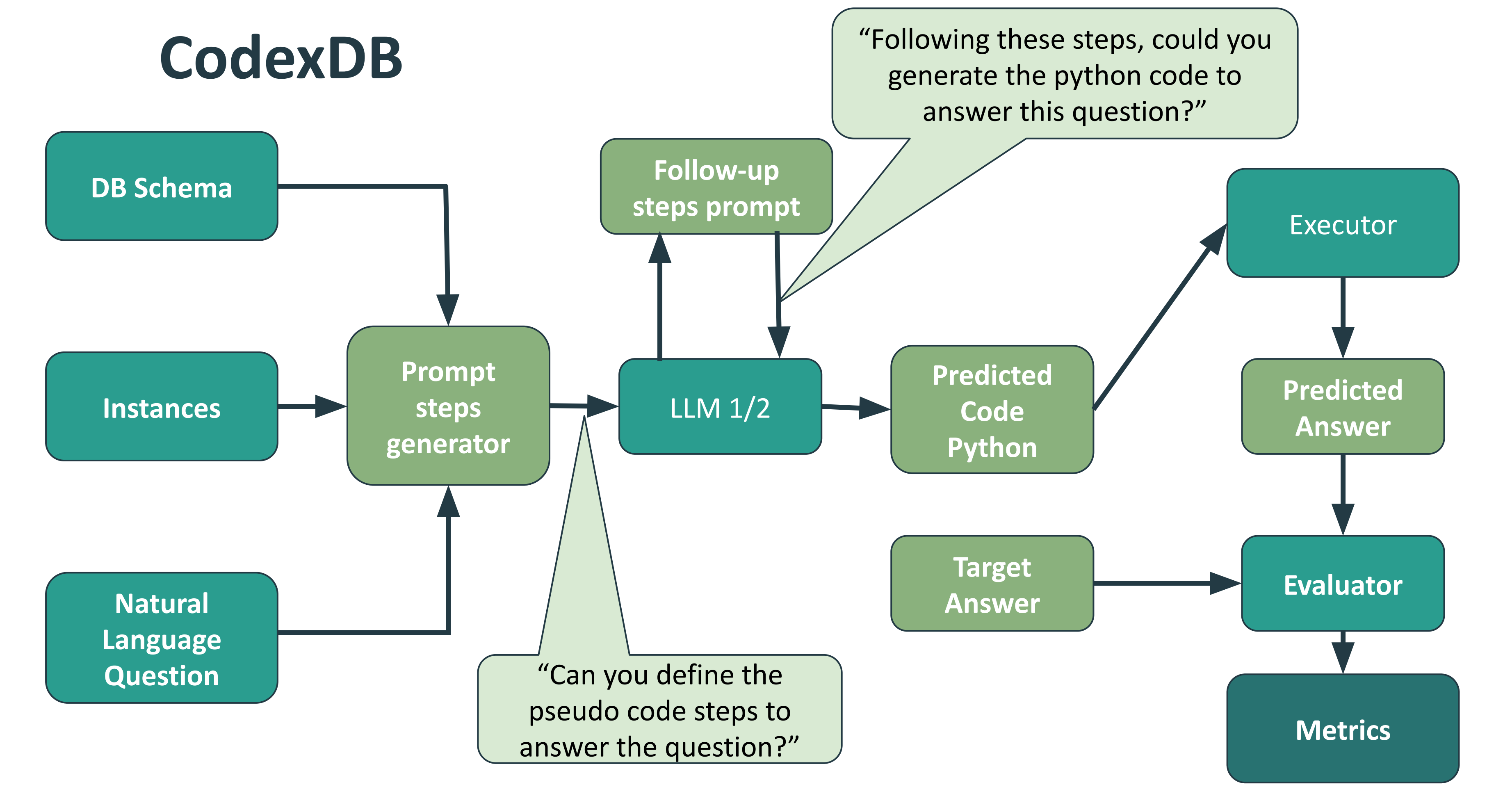
Quality code metrics:

Cyclomatic Complexity, Halstead Measures, Maintainability Index

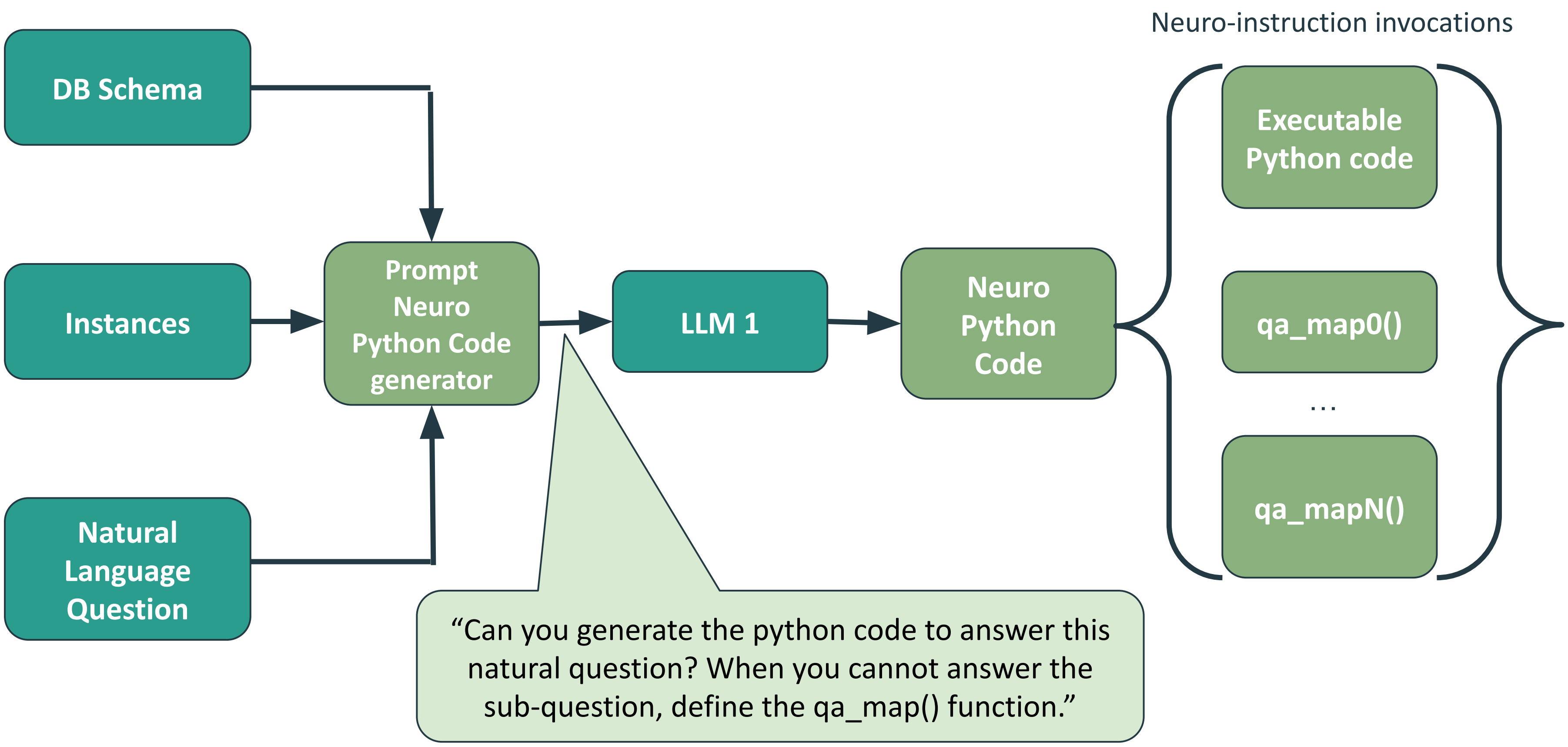


Vanilla Code Python

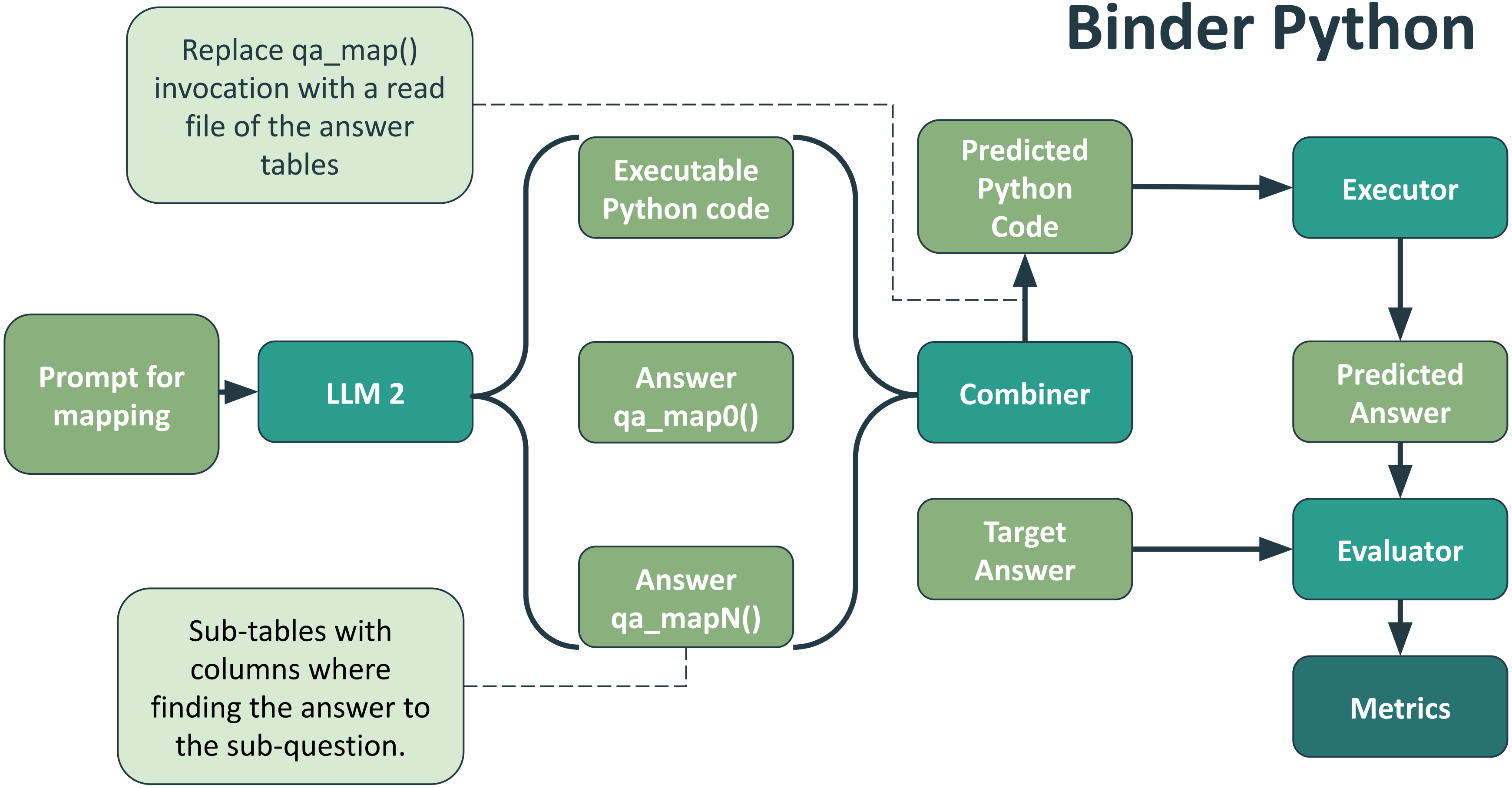



CodexDB
DB Schema
Instances
Natural Language Question
Prompt steps generator
“Can you define the pseudo code steps to answer the question?”
LLM 1/2
Follow-up steps prompt
“Following these steps, could you generate the python code to answer this question?”
Predicted Code Python
Executor
Predicted Answer
Target Answer
Evaluator
Metrics

Binder Python

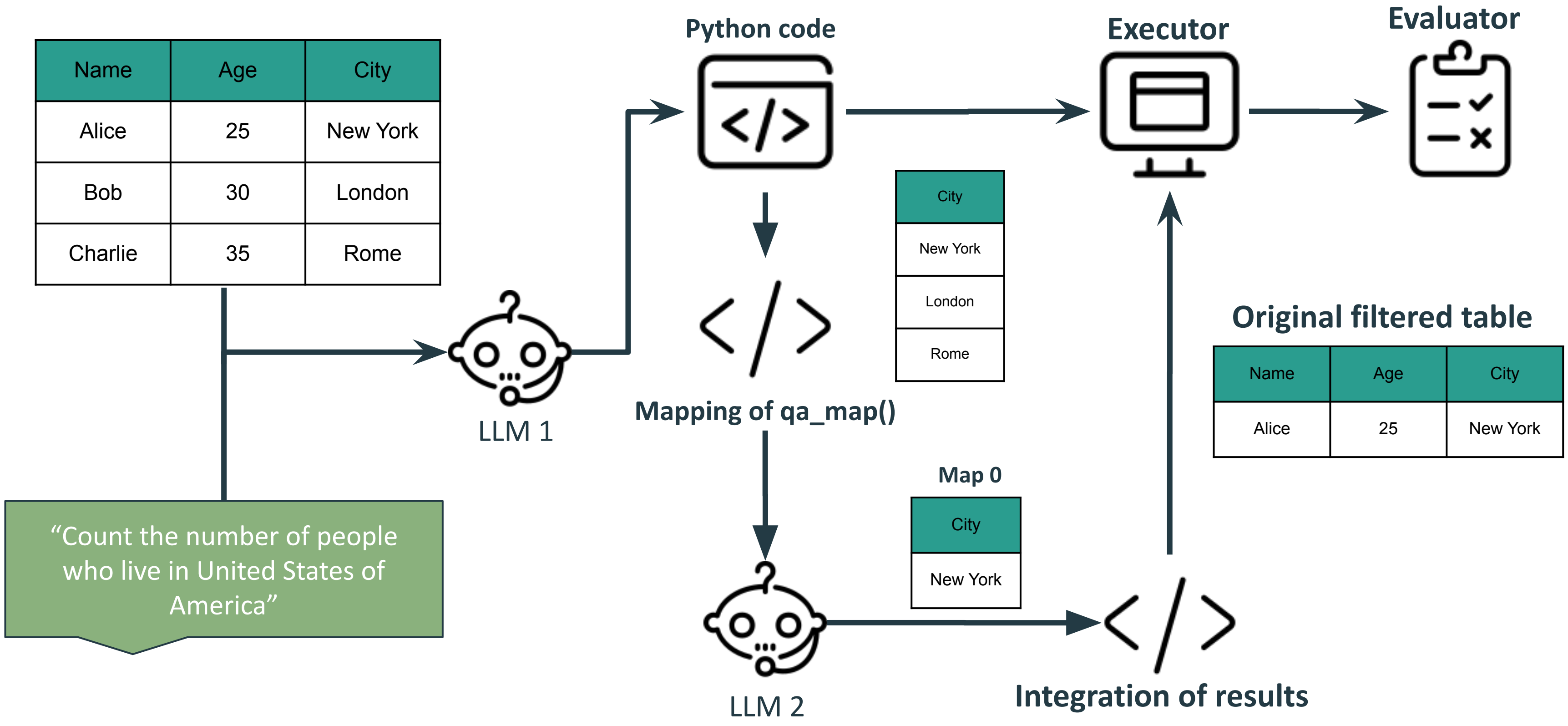


Binder Python



Binder Python example

qa_map('table_default.csv', 'which cities are in the United States', ['City'])



Binder Python example

Neuro Python Code generated

```
# Use qa_map to filter the table
for cities in the United States
filtered_table =
qa_map('table_default.csv', 'which
cities are in the United States',
['City'])

# Count the number of people who
live in the United States
count_usa = len(filtered_table)

# Print the result
print([[count_usa]])
```

Code after results integrations

```
#Use qa_map to filter the table
for cities in the United States
filtered_table =
pd.read_csv('map0.csv')

# Count the number of people who
live in the United States
count_usa = len(filtered_table)

# Print the result
print([[count_usa]])
```

Metrics: Evaluating Model Performance

To **provide** a **comprehensive analysis** of the **performance of our models**, we used **two different classes of metrics**:

QATCH Metrics:
Unified evaluation metrics used across QA, Text-to-SQL, and Text-to-Code, measuring both **semantic correctness** and **structural fidelity** of the model outputs.

Code Metrics:
A custom suite designed for Text-to-Code, evaluating the **complexity**, **maintainability**, and **readability** of generated Python code.

Code Metrics

Cyclomatic Complexity

Measures the number of independent "paths" that can be taken in the code execution flow. It is related to conditional logic (if, for, while, try, etc.).

Components

E = arcs,
N = nodes,
P = connected components

$$CC = E - N + 2P$$

Returns a list of values representing the complexity for each independent "path"

Code Metrics

Halstead Metrics

Measures the cognitive complexity of code based on the operators and operands used. It is useful for estimating how difficult it is to understand or maintain a program.

Components

n1: distinct number of operands
n2: distinct number of operands
N1: total number of operands
N2: total number of operands

Volume (V) = $(N1 + N2) \times \log_2(n1 + n2)$
→ amount of information

Difficulty (D) = $(n1 / 2) \times (N2 / n2)$
→ difficulty of comprehension

Effort (E) = $D \times V$
→ estimated mental effort

Code Metrics

Maintainability Index

A composite index that measures how easy a piece of code is to maintain.

Components

CC = Cyclomatic Complexity

HV = Halstead Volume

NLC = Number of Lines of Code

$$MI = 171 - 5.2 * \ln(HV) - 0.23 * CC - 16.2 * \ln(NLC)$$

QATCH Metrics

🧮 **Cell Precision**	**Percentage** of **predicted table cells** that **are correct**.
🎯 **Cell Recall**	**Percentage** of **ground-truth table cells** that were **successfully retrieved.**
🧩 **Tuple Constraint**	**Exact match on schema**, **cardinality**, and **cell values** (1 if identical, 0 otherwise).
📊 **Tuple Cardinality**	**Ratio** of **predicted** to **ground-truth tuple counts**.
🔢 **Tuple Order**	**Correlation** between **predicted vs. true tuple ordering** (for ORDER BY queries).

Results

Experiments on **three distinct benchmark**

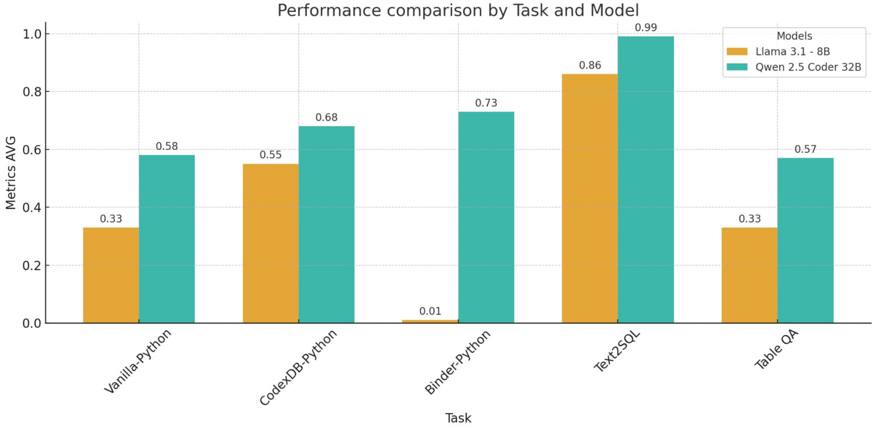
- SPIDER 1.0
- BEAVER
- WIKITABLE QUESTIONS

Two different Large Language Models

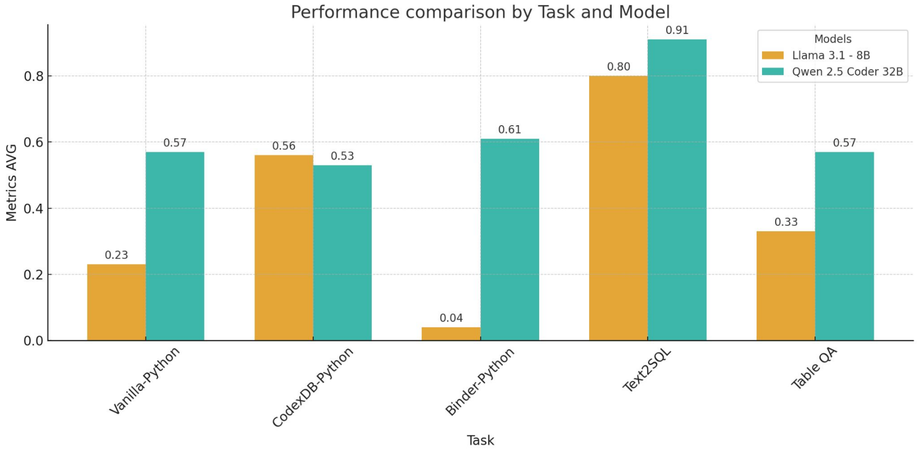
- LLaMA 3.1 8B
- Qwen 2.5 coder 32B

Preliminary Tests

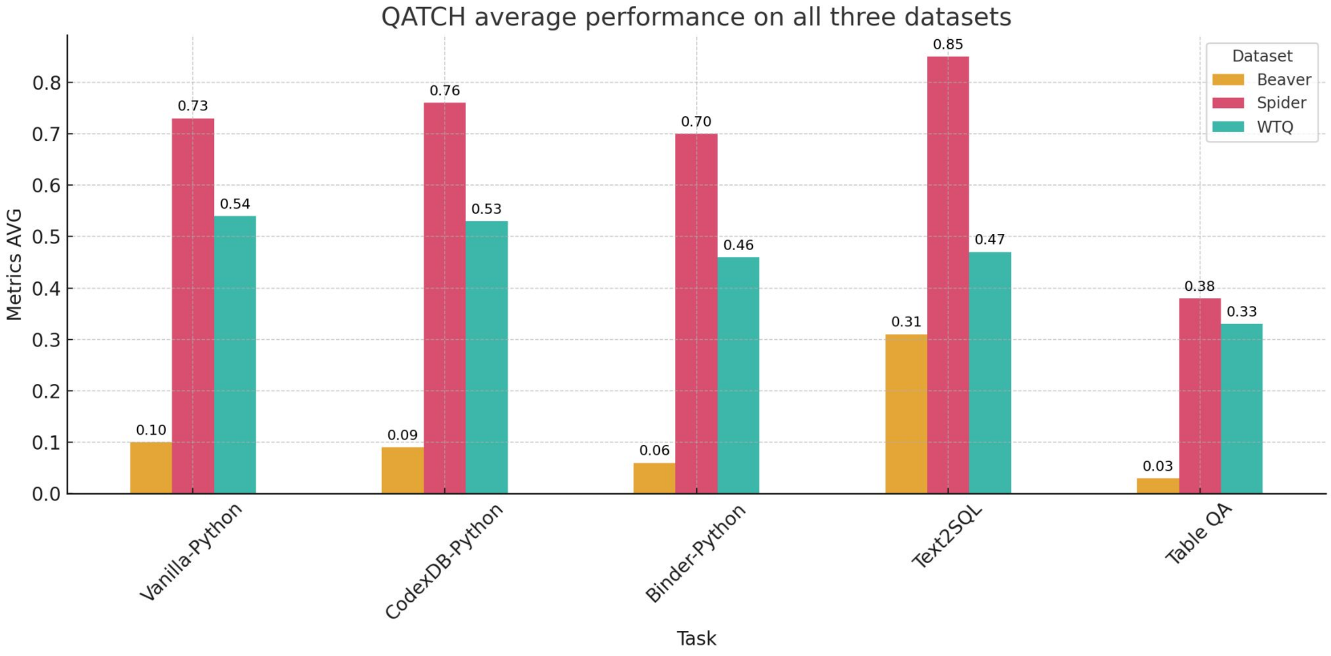
[Non-proprietary] Spider avg metrics (Concert Singer)



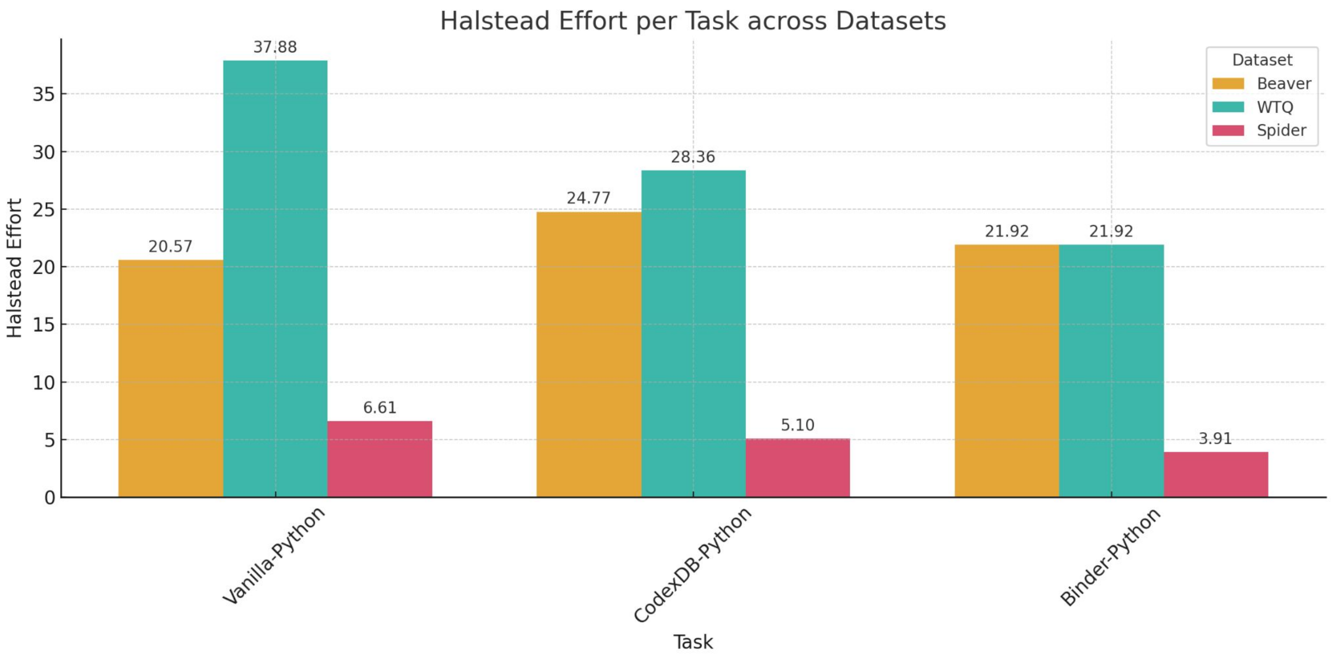
[Proprietary] BEAVER avg metrics (TIME QUARTERS)



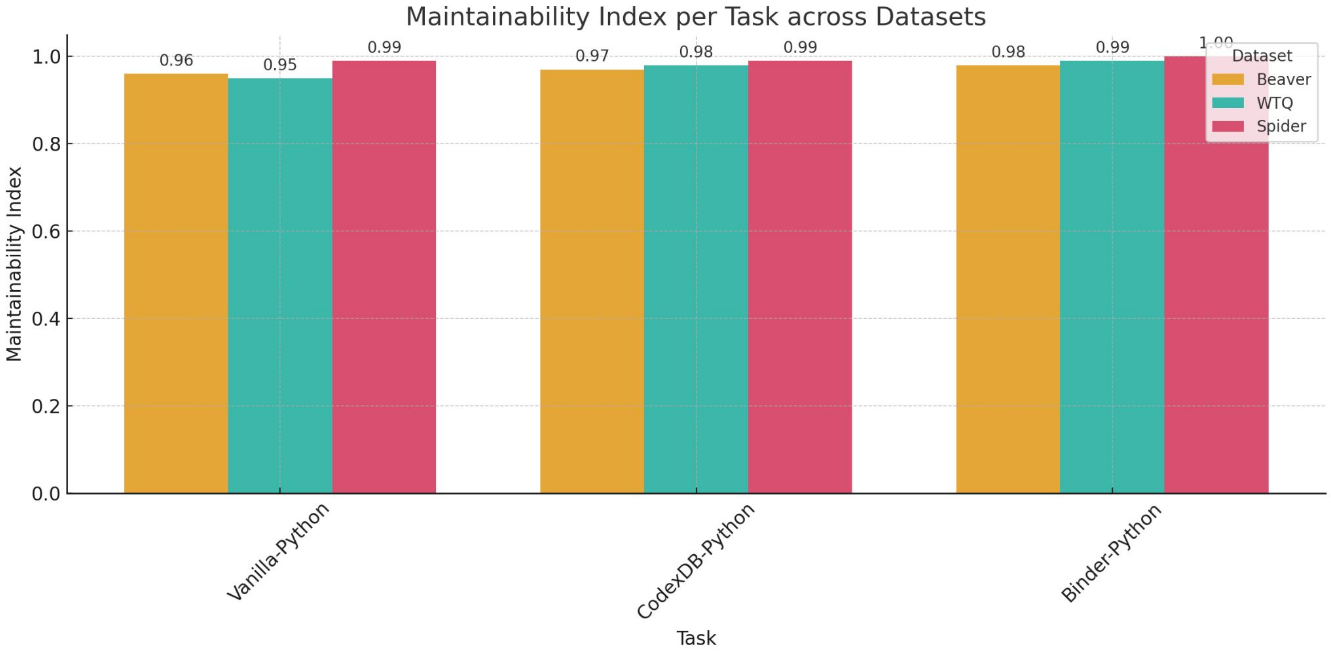
QATCH Results



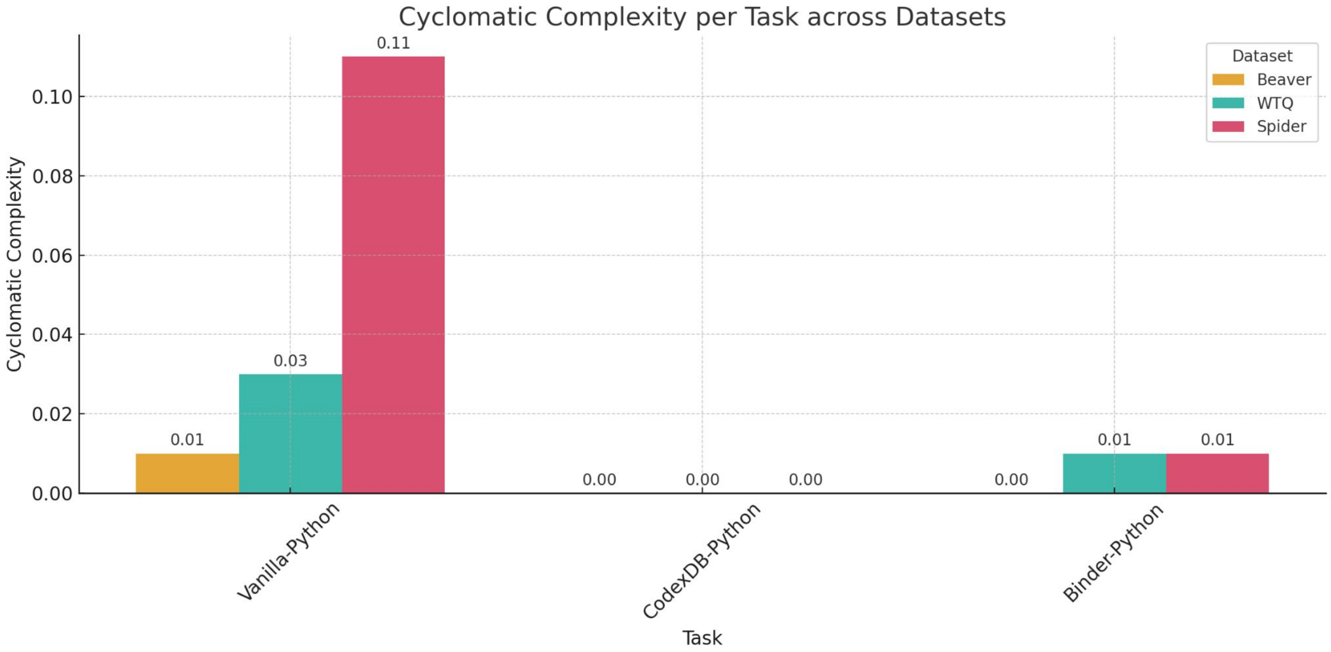
Halstead (effort) Results



Maintainability Index Results



Cyclomatic Complexity Results



Conclusions

Some models show stronger **task-specific reasoning**, especially in structured domains.

LLMs prefer **simpler tasks with clearer structure**, especially when schema info is limited.

Text2SQL is stronger on SQL-centric tasks, while **Text2Code** excels in reasoning intensive ones.

There is **room for improvement** in handling complex datasets and ambiguous queries.

Model performance varies depending on dataset type (e.g., BEAVER vs. SPIDER).

Prompt design and instance diversity play a critical role in code generation quality.

Thanks for your attention ;)

Questions?

Future works

Enhance existing pipelines
Refine Vanilla, CodexDB, and Binder to handle prompt sensitivity and memory issues.

Explore new Text2Code paradigms
Explore methods like chain-of-thought and program-by-example for better reasoning.

Automatic schema induction
Develop automatic schema induction for semi-structured tables like WTQ.

Benchmark on no–SQL-centric datasets
Evaluate performance on datasets like TabFact or HybridQA.

Evaluate different model types
Use code-specialized and multimodal models to assess task suitability.

Scale Table QA to full datasets
Implement full-dataset QA with retrieval and streaming for large tables.