Stanford CS224v Course

Conversational Virtual Assistants with Deep Learning

# Lecture 7

# SUQL: Structured/Unstructured Query Language

Monica Lam & Shicheng Liu

# Summary from Lecture 6

- **Many questions require information from hybrid data sources**
- **Prior work**
  - **Structure OR Text: is inadequate**
    - Binary classifier up front (SK-TOD, 2023)
    - Pick afterwards (Stanford Chirpy Cardinal, 2021)
  - **Different approaches to combine structures and free-text**
    - Structures → Text: Linearization (one hop)
    - Hybrid: Retrieve from both and combine (one hop each)

We need to compose answers from both sources arbitrarily!

# Goals for Handling Hybrid Data

1. **Maximize available information retrieval capability**
   - Keep text as text and KBs as KBs
     - Combine the best of two worlds!
   - Converting KB → Text, or Text → KB
     - Worse when choosing the lowest denominator
2. **Hybrid multi-hop: Compose IR and KB accesses arbitrarily** NEW!
   - Efficient searching of free text (using embedded similiarity)
   - Relational algebra operations (aggregate, rank, …)

# QUIZ

## HOW DO WE SUPPORT FULL COMPOSITIONALITY?

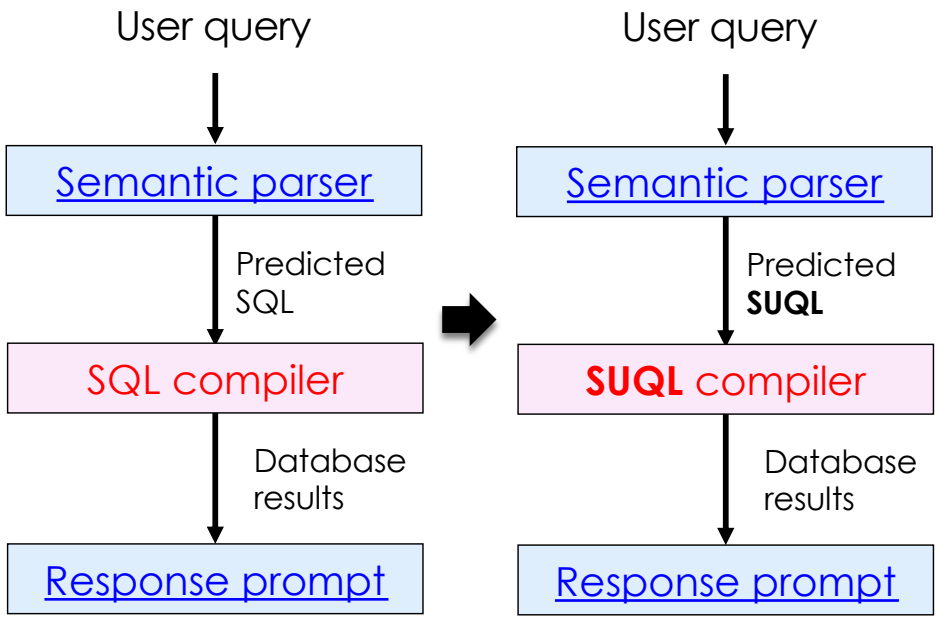
# Invent the SUQL Language

- **SUQL (Structured & Unstructured Query Language)**
  - Extends SQL to include free-text operations
  - Leverages LLM familiarity
    with the most popular DB query language
- **SUQL-based Genie Framework**
  - Developer only needs to provide
    SQL schema + free-text data

# Reduces Complexity for Neural Model

SUQL: A high-level language
**– *domain agnostic***

- Decompose implementation into:
  - *LLM-based semantic parser*
  - Algorithmic **full-query** optimizations in the ***optimizing*** *compiler*



# Genie Agent Framework Inputs (Yelp Example)

DB Schema

```
CREATE TABLE restaurants (
    id TEXT PRIMARY KEY,
    location TEXT,
    address TEXT,
    cuisines ENUM[],
    rating NUMERIC(2,1),
    ...,
    popular_dishes TEXT[],
    reviews TEXT[]
...);
```

NEW!
NEW!

Free-Text
SUQL framework will create the vector store for IR

```
--
User: Show me a family-friendly restaurant that has burgers
Target: SELECT *, summary(reviews), answer(reviews, 'is this restaurant family-friendly?') FROM restaurants WHERE answer(reviews, 'do you find this restaurant to be family-friendly?') = 'Yes' AND answer(popular_dishes, 'does this restaurant serve burgers') = 'Yes' LIMIT 1;
--
User: Find me a place with pasta.
Target: SELECT *, summary(reviews) FROM restaurants WHERE answer(popular_dishes, 'does this restaurant serve pasta') = 'Yes' LIMIT 1;
--
...
```

Few-shot examples (optional)

# Lecture Goals

- SUQL Language – Design and Rationale
- SUQL Implementation – Semantic parser & Compiler
- Evaluation in a real life app: Yelp
- Evaluation with a large dataset: HybridQA
  (with an improvement)

# SUQL Free-Text Support

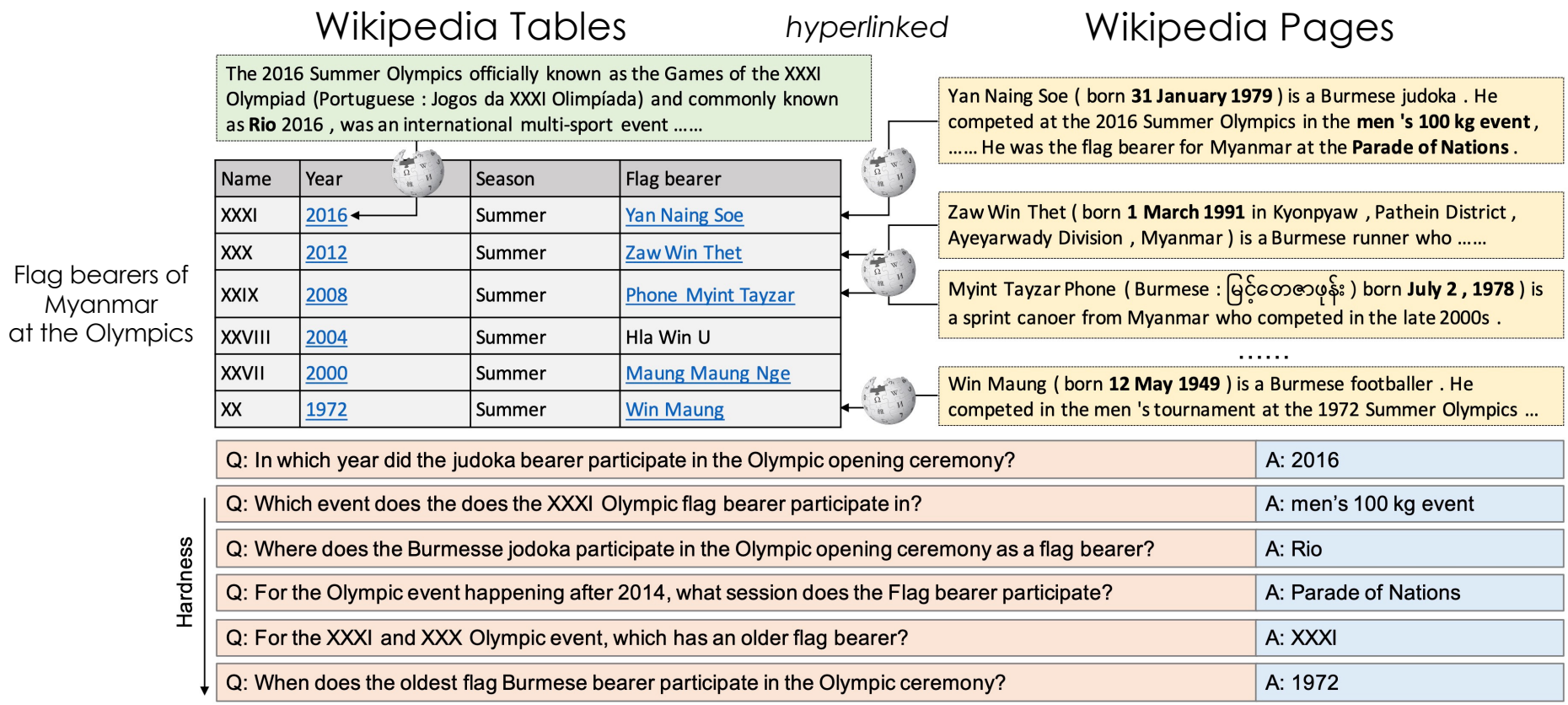
- Add free-text primitives into SQL, implemented with IR & LLM
- Two functions: summary, answer

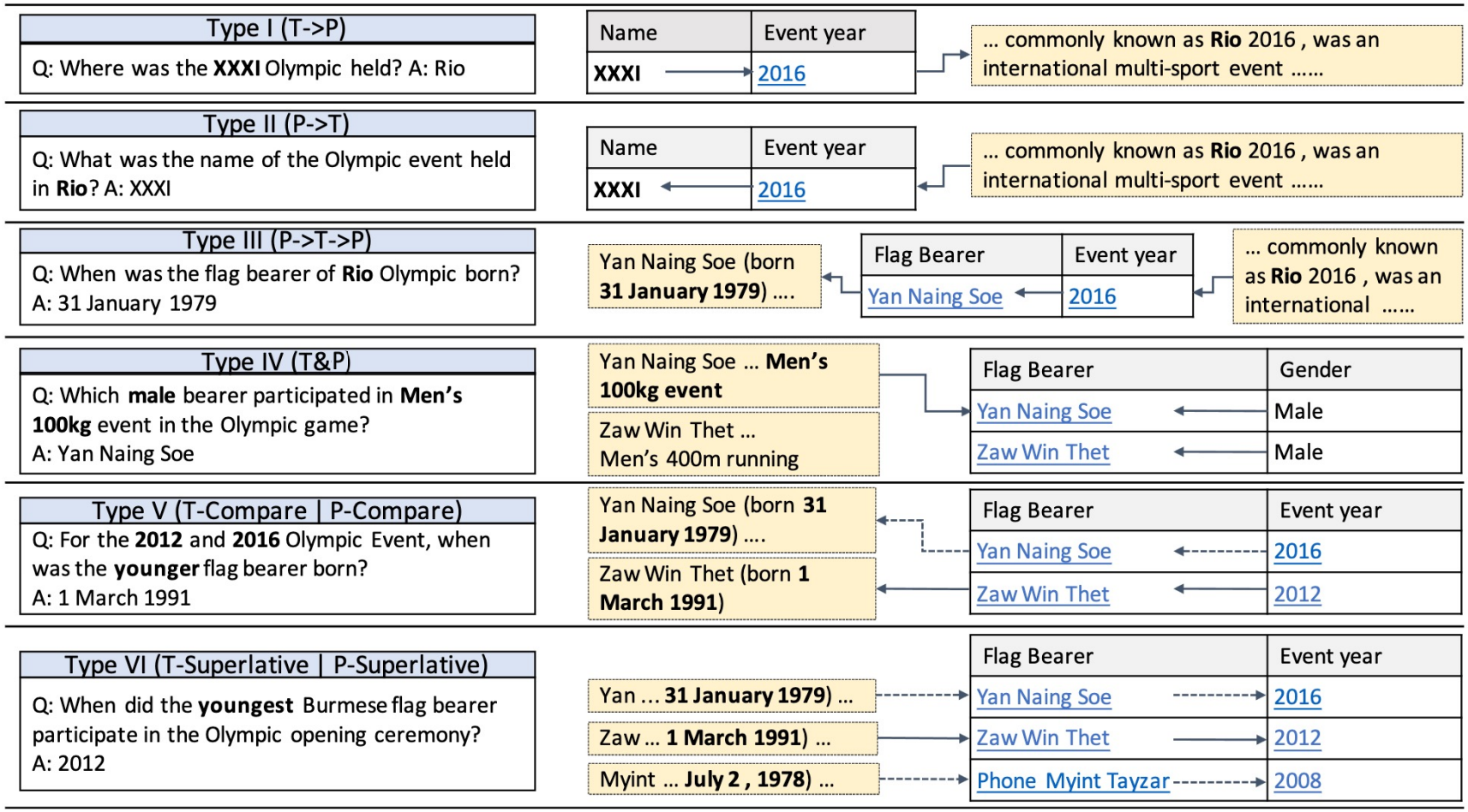
"I want a family-friendly restaurant in Palo Alto"

```
SELECT *, summary(reviews) FROM restaurants
    WHERE location = 'Palo Alto'
AND answer(reviews, 'is it a family friendly restaurant') = 'Yes' LIMIT 1;
```

Quiz: How do you implement summary and answer?

# HybridQA Dataset






T: Table
P: Passage

Figure 3: Illustration of different types of multi-hop questions.

# HybridQA Questions in SUQL

```
CREATE TABLE Flag (
"Name" TEXT,
"Flag Bearer" TEXT,
"Flag Bearer_Info" TEXT[],
"Gender" TEXT,
"Event year" TEXT,
"Event year_Info" TEXT[]);
```

T: Table
P: Paragraph

**Type I (T->P)**

```
SELECT answer("Event year_Info",
                    'where is this event held?')
          FROM "Flag" WHERE "Name" = 'XXXI'
```

# HybridQA Questions in SUQL

**Type II (P->T)**

```
CREATE TABLE Flag (
"Name" TEXT,
"Flag Bearer" TEXT,
"Flag Bearer_Info" TEXT[],
"Gender" TEXT,
"Event year" TEXT,
"Event year_Info" TEXT[]);
```

```
SELECT "Name" FROM "Flag"
WHERE answer("Event year_Info",
        'where is this event held?') = 'Rio'
```

T: Table
P: Paragraph

# HybridQA Questions in SUQL

**Type III (P->T->P)**

```
CREATE TABLE Flag (
"Name" TEXT,
"Flag Bearer" TEXT,
"Flag Bearer_Info" TEXT[],
"Gender" TEXT,
"Event year" TEXT,
"Event year_Info" TEXT[]);
```

T: Table
P: Paragraph

```
SELECT answer("Flag Bearer_Info",
                    'when is this person born?')
FROM "Flag"
WHERE answer("Event year_Info", 'where is this event
held?') = 'Rio'
```

# HybridQA Questions in SUQL

**Type IV (T&P)**

```
CREATE TABLE Flag (
"Name" TEXT,
"Flag Bearer" TEXT,
"Flag Bearer_Info" TEXT[],
"Gender" TEXT,
"Event year" TEXT,
"Event year_Info" TEXT[]);
```

T: Table
P: Paragraph

```
SELECT "Flag Bearer" FROM "Flag" WHERE
"Gender" = 'Male' AND
answer("Flag Bearer_Info",
        'what event did this person participate in?')
        = "Men's 100kg event"
```

# HybridQA Questions in SUQL

**Type V (T-Compare | P-Compare)**

```
CREATE TABLE Flag (
"Name" TEXT,
"Flag Bearer" TEXT,
"Flag Bearer_Info" TEXT[],
"Gender" TEXT,
"Event year" TEXT,
"Event year_Info" TEXT[]);
```

T: Table
P: Paragraph

```
SELECT MAX
(answer("Flag Bearer_Info",
            'when is this person born?')::date)
FROM "Flag" WHERE "Event year" IN ('2016', '2012')
```

# HybridQA Questions in SUQL

**Type VI (T-Superlative | P-Superlative)**

```
CREATE TABLE Flag (
"Name" TEXT,
"Flag Bearer" TEXT,
"Flag Bearer_Info" TEXT[],
"Gender" TEXT,
"Event year" TEXT,
"Event year_Info" TEXT[]);
```

T: Table
P: Paragraph

```
SELECT "Event year" FROM "Flag" ORDER BY
answer("Flag Bearer_Info",
        'when is this person born?')::date
DESC LIMIT 1;
```

# At-A-Glance: HybridQA Questions in SUQL

```
CREATE TABLE Flag (
"Name" TEXT,
"Flag Bearer" TEXT,
"Flag Bearer_Info" TEXT[],
"Gender" TEXT,
"Event year" TEXT,
"Event year_Info" TEXT[]);
```

**Type I (T->P)**
Where was the XXXI Olympic held?

```
SELECT
answer("Event year_Info",
        'where is this event held?')
FROM "Flag" WHERE "Name" = 'XXXI'
```

**Type II (P->T)** What was the name of the Olympic event held in Rio?

```
SELECT "Name" FROM "Flag"
WHERE answer("Event year_Info",
'where is this event held?') = 'Rio'
```

**Type III (P->T->P)** When was the flag bearer of Rio Olympic born?

```
SELECT answer("Flag Bearer_Info",
'when is this person born?')
FROM "Flag"
WHERE answer("Event year_Info",
'where is this event held?') = 'Rio'
```

**Type IV (T&P)** Which **male** bearer participated in **Men's 100kg** event in the Olympic game?

```
SELECT "Flag Bearer" FROM "Flag"
WHERE "Gender" = 'Male' AND
answer("Flag Bearer_Info",
'what event did this person
participate in?')
= "Men's 100kg event"
```

**Type V (T-Compare | P-Compare)** For the **2012** and 2016 Olympic Event, when was the younger flag bearer born?

```
SELECT MAX
(answer("Flag Bearer_Info",
  'when is this person born?')::date)
FROM "Flag"
WHERE "Event year" IN ('2016', '2012')
```

**Type VI (T-Superlative | P-Superlative)** When did the **youngest** Burmese flag bearer participate in the Olympic opening ceremony?

```
SELECT "Event year" FROM "Flag"
ORDER BY answer("Flag Bearer_Info",
'when is this person born?')::date
DESC LIMIT 1;
```

Quiz: Where can 'answer' be used in a query?

# Conversational Example: Restaurant

**Restaurants**

Do you have a recommendation for a first date restaurant in Palo Alto?

We're thinking sushi but not sure what's good around here.

```
SELECT *, summary(reviews) FROM restaurants
WHERE 'sushi' = ANY (cuisines) AND location = 'Palo Alto' AND rating >= 4.0
AND answer(reviews, 'is this restaurant good for a first date?') = 'Yes'
ORDER BY num_reviews DESC LIMIT 1;
```

# Conversational Example: Shopping

```
CREATE TABLE laptop (
"ram" int,
…,
"about" TEXT,
“description" TEXT[],
"reviews” TEXT);
```

**Laptops**

I need a laptop with a Thunderbolt 3 port and at least 16GB RAM for my workstation setup.

↓

```
SELECT *, summary(reviews) FROM laptops
WHERE ram >= 16 AND
    (answer(about, 'does this laptop have Thunderbolt 3?') = 'Yes’
OR answer(description, 'does this laptop have Thunderbolt 3?') = 'Yes’)
LIMIT 3;
```

*It is often unclear where the answer is in the text, useful/easy to search for all possibly relevant text fields.*

# QUIZ

## WHAT CAN WE USE SUQL FOR?

# Advantages of SUQL

- **Formal representation combining IR and DB query**
  - Succinct, expressive, domain independence
  - Full compositionality: combining IR and relational algebra
- **Interpretability**
  - We can read the query back
    so users know what question is being answered
  - Quiz: Is the answer always correct given a formal query?

- **SUQL is a high-level programming language**
  - Optimization issues handled by compilers, abstracted from programmers

# SUQL is New

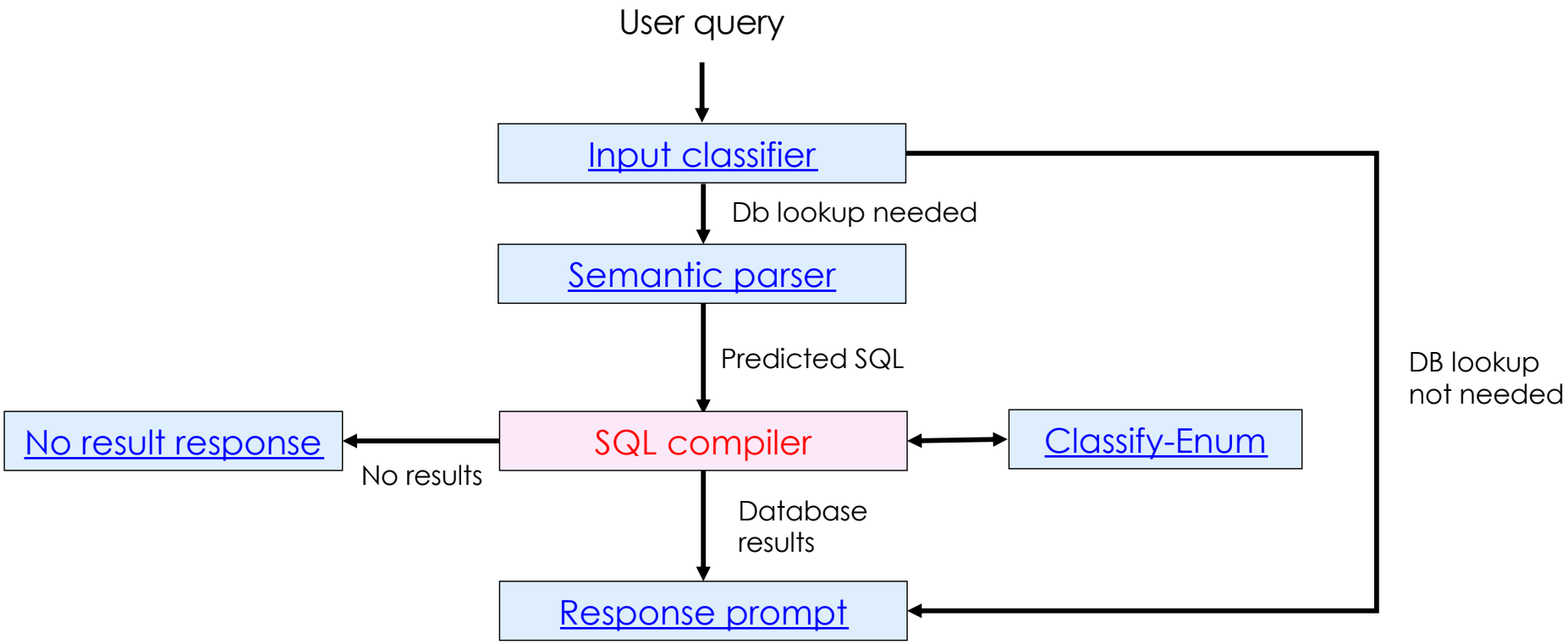
- How to evaluate SUQL as a design?

- What is the accuracy?
  - Semantic parser accuracy
  - Execution accuracy
- What is the speed?

# Lecture Goals

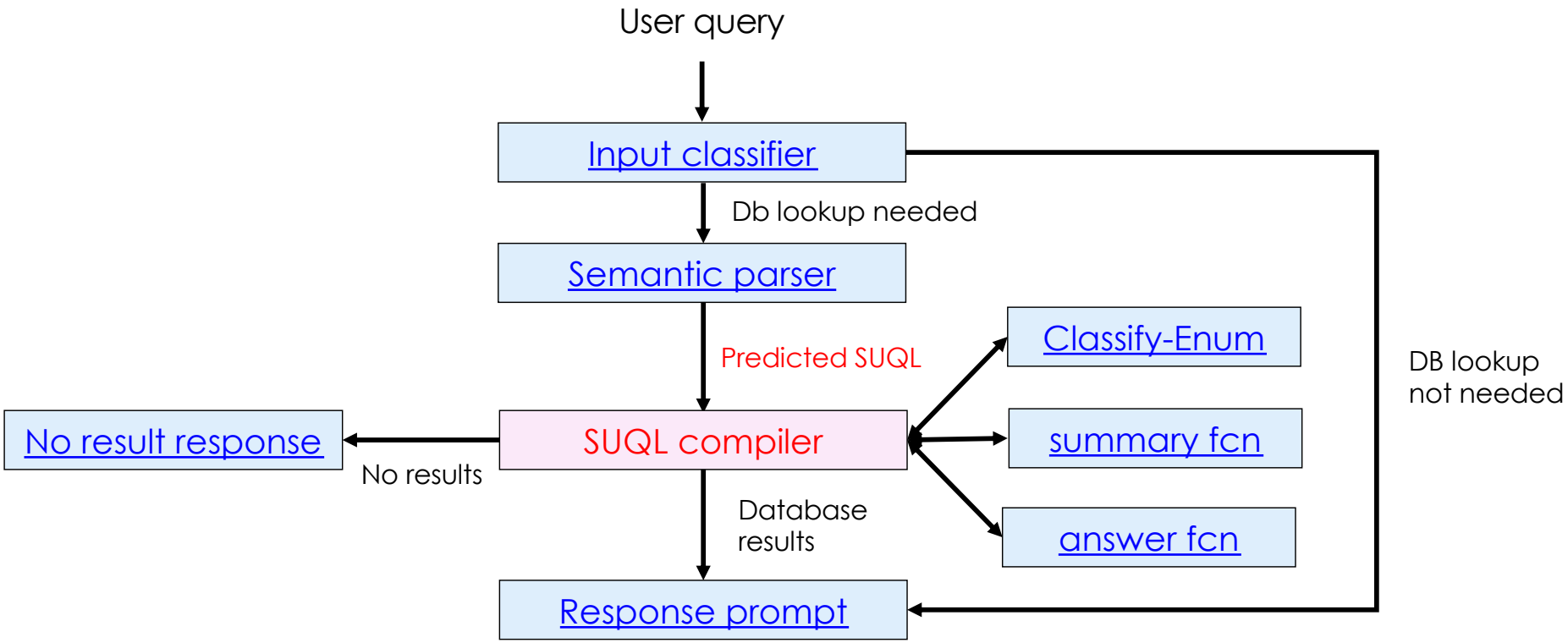
- SUQL Language – Design and Rationale
- **SUQL Implementation – Semantic parser & Compiler**
- Evaluation in a real life app: Yelp
- Evaluation with a large dataset: HybridQA (with an improvement)

# Agent Design for SQL




Click the links to see the prompts (written in jinja syntax)

# Agent Design: SQL Updated with SUQL




7 prompts with few-shot examples. Templates written in jinja syntax

# SUQL:
# High-Level Programming Language

The Compiler can Implement Optimizations
Hiding details from Users

# SUQL Compiler

- SQL can run SUQL programs without modification
  - summary / answer are just external functions

- But it is slow

# Example

"I want a family-friendly restaurant in Palo Alto"

```
SELECT *, summary(reviews) FROM restaurants
WHERE  answer(reviews, 'is it a family friendly restaurant') = 'Yes'
       and  location = 'Palo Alto'
```

- Consider a naïve implementation
  - Retrieve each review from the table
  - For each review, call LLM to answer the question
  - Filter on location
  - Summarize all the returned reviews
  - Report the first few

Quiz: How would you optimize the query?

# 1. Order Filtering

answer(reviews, ‘is it a family friendly restaurant’) = ‘Yes’
AND
and location = 'Palo Alto’

- Execution of structured predicates is much cheaper
- Always execute structured predicates first

# 2. Return Only Necessary Results

answer(reviews, 'is it a family friendly restaurant') = 'Yes'

- For applications such as recommendation, it is not necessary to return all the answers
- IR uses embedding model (vector similarity) to return top candidates
- Return only top results to LLM-based answer functions

# 3. Lazy Evaluation

answer(reviews, ‘is it a family friendly restaurant’) = ‘Yes’
AND
answer(reviews, ’is it easy to park’) = ‘Yes’
AND
and location = 'Palo Alto’ LIMIT 1

- Lazy evaluation: Evaluate only when the result is needed
- Stop calling answer as soon as LIMIT 1 is reached

# SUQL Compiler Implementation

- SUQL is implemented by modifying the SQL compiler
- To support arbitrary composition of SQL and summary/answer statements
  - Process the syntax tree of SQL recursively
    - Start from the bottom node with no sub-queries
    - And move to the top

# SUQL Compiler Implementation (cont.)

- For each SELECT statement with answer in it
  Apply optimizations to the SELECT statement
  - Store the processed results in a temporary table *temp*
  - Substitute this statement with *SELECT from temp*

- SQL compiler handles the final processing

# Compiler Optimizations

- Possible only because we are optimizing across the whole query

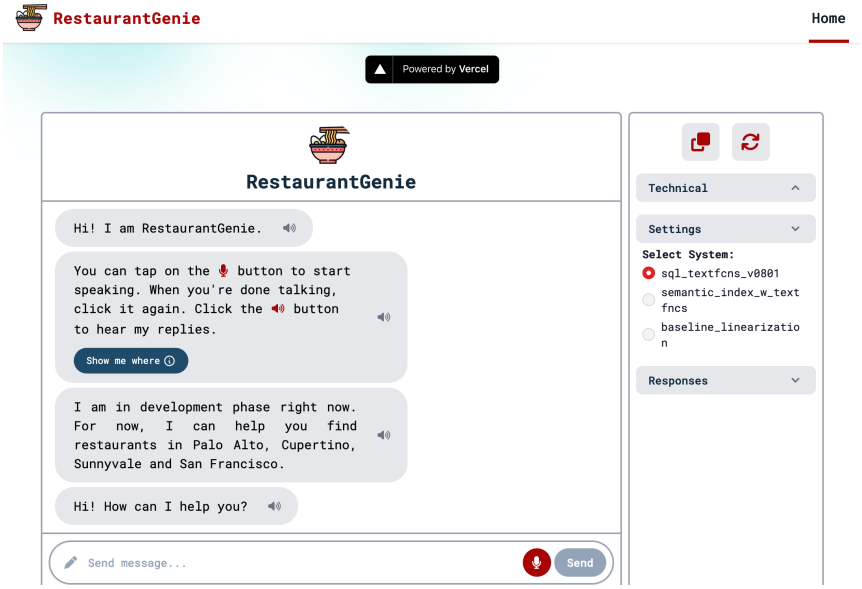
Quiz: Is the optimization necessary?

Quiz: What if we use an agent
that uses "function calling" to call IR and query databases separately?

# Lecture Goals

- SUQL Language – Design and Rationale
- SUQL Implementation – Semantic parser & Compiler
- **Evaluation in a real life app: Yelp**
- Evaluation with a large dataset: HybridQA (with an improvement)

# Experiment




A real, large dataset

- Yelp on San Francisco, Palo Alto, Cupertino, Sunnyvale
- Scraped reviews and popular dishes information

Components

- LLM: gpt-3.5-turbo
- Information retrieval on text fields
  - indexed with Coco-DR
- A new optimizing SUQL compiler

COCO-DR reference: https://arxiv.org/abs/2210.15212

# Evaluation (in Restaurants)

- **We solicit user queries via crowdsourcing on Prolific.**
  - We do not disclose to the workers what fields are available in the database.

- **The Yelp Dataset:**
  1. Just Q&A: 100 crowdsourced questions about restaurants.
  2. Conversational: 20 conversations with 96 turns

# The Need for Hybrid QA in Real Life?

| | Single-turn | Conversation |
|---|---|---|
| Structured-only | 45 | 37 (59%) |
| Combination | 55 | 25 (41%) |
| Total | 100 | 62 |

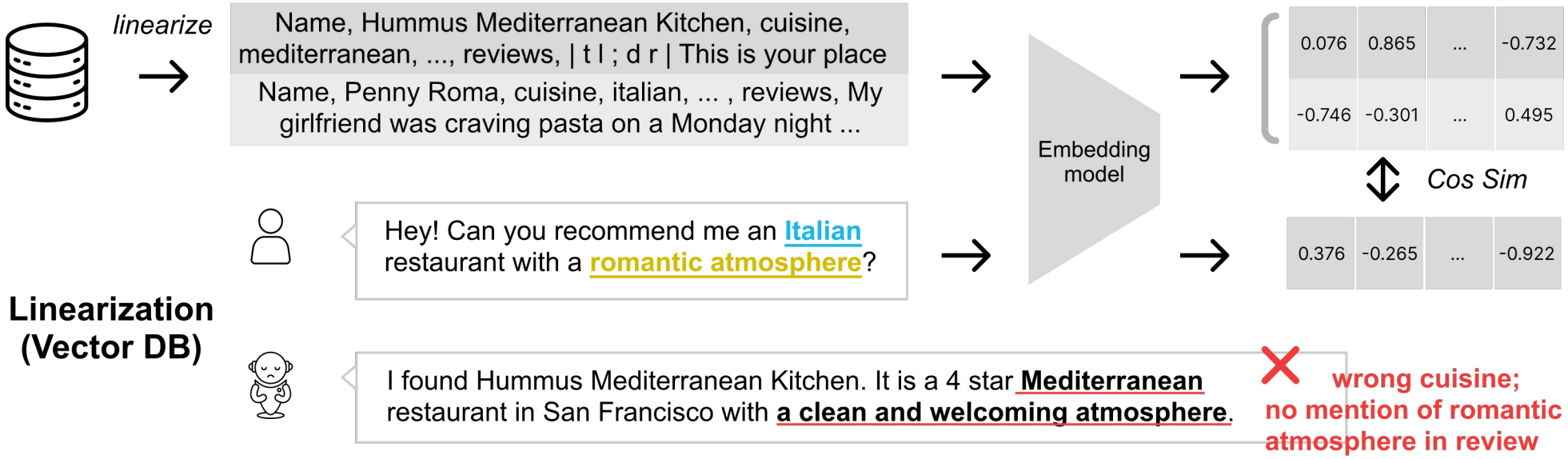
# Evaluation (in Restaurants)

Quiz: how do you evaluate its accuracy?

- In real-world tasks: Difficult to label the gold answers:
  - For each query, there are many possible "gold restaurants"

- We manually inspect whether restaurants returned satisfy all criteria
- Calculate Query *precision*: #correct results / #results

# Baseline comparison: Linearize + Information Retrieval




Embedding based matches are poor at multi-criteria queries

# Using SUQL

Hey! Can you recommend me an **Italian** restaurant with a **romantic atmosphere**?

↓ *Semantic Parser*

```
SELECT *, summary(reviews) FROM restaurants
WHERE 'italian' = ANY (cuisines) AND
answer(reviews, 'is this restaurant romantic?') = 'Yes'  LIMIT 1;
```

*SUQL Compiler* →

I found Penny Roma, which has a 4.0 rating on our database and offers a variety of **Italian** dishes. Overall, the atmosphere is described as delightful, authentic, and **perfect for a date spot**.

# Evaluation

| | Single-turn | Conversational |
|---|---|---|
| Linearization @ 1 | 57.0 % | 63.4 % |
| Linearization @ 3 | 49.7 % | 61.9 % |
| SUQL | **93.8 %** | **90.3 %** |

Table 5: Turn accuracy measurement on linearized system versus SUQL system.

Quiz: how can you get incorrect results with semantic parsing on SUQL?

# What About Recall?

- Are all the answers found?
  - Incomplete
  - Not at all

← Quiz: Is this OK?

# Defensive Programming:
# What if the Semantic Parser is Wrong?

Always verbalize the user query

Allows the user to spot mistakes in the answer

Even when it fails, so users can adjust the query.

> I searched for 5-star restaurants in Sunnyvale that serve kids food.
>
> Unfortunately, I couldn't find any search results.
>
> Is there anything else I can help you with?

# Error Analysis:
# What Happened When No Answers Were Returned?

14 false negatives on our single-turn set (100 questions)

- Parsing: 2 errors (one syntactic, one enum field confusion)
- Query evaluation
  - Structured: ← Can be improved
    - 4 unsupported location service
    - 2 opening hours errors
    - 2 dishes: searched in popular dishes, but results are in reviews
  - Free-text: 4 false negatives from 'answer' (ChatGPT)

# User Feedback - Positive

- *There's actually nothing I didn't like about this chatbot.*
- *I would honestly use this chatbot on a regular basis if it were available to the public*
- *I liked that the chatbot was fast in responses and it gave very detailed responses and I hardly had any questions about a restaurant after the option was given.*
- *Shocked at how good the restaurant suggestions were. I even asked for something with better prices and got that too. Now I'm hungry. I asked to define a cuisine style and it was able to do that.*

# User Feedback - Negative

- *Occasional slowness of the chatbot*
- *It didn't provide any links or pictures*
- *It did not sound friendly and sometimes the responses were too long.*
- *Bullet point outputs would be much more helpful.*

# Lecture Goals

- SUQL Language – Design and Rationale
- SUQL Implementation – Semantic parser & Compiler
- Evaluation in a real life app: Yelp
- **Evaluation with a large dataset: HybridQA (with an improvement)**

# Hybrid QA Dataset

| Dataset | Size | #Docs | KB/ Table | Multi-Hop |
|---|---|---|---|---|
| WebQuestions | 5.8K | no | yes | yes |
| WebQSP | 4.7K | no | yes | yes |
| WebQComplex | 34K | no | yes | yes |
| MetaQA | 400k | no | yes | yes |
| WikiTableQA | 22K | no | yes | yes |
| SQuAD-v1 | 107K | 1 | no | no |
| DROP | 99K | 1 | no | yes |
| TriviaQA | 95K | >1 | no | no |
| HotpotQA | 112K | >1 | no | yes |
| Natural-QA | 300K | >1 | no | yes |
| HYBRIDQA | 70K | >>1 | yes | yes |

# HybridQA Dataset: Based on Wikipedia

- 13000 tasks on Amazon Turk:
  - Given one crawled Wikipedia table + its hyperlinked passages.
  - Write 6 questions & answers.
  - Each task takes 12 minutes on average, and payment is $2.3 U.S.
- 5 CS graduate students to decide on the acceptance.

| #Table | #Row/#Column | #Cell | #Links/Table |
|---|---|---|---|
| 13,000 | 15.7/4.4 | 70 | 44 |
| #Passage | #Words/Passage | #Ques | #Words/Ques |
| 293,269 | 103 | 69,611 | 18.9 |

# HybridQA Dataset

- **Question rely on both tabular and textual information.**
- **Table reasoning step specifically includes**
  - filter rows based on =, >, <: “the XXXI Olympic event”
  - superlative operation over a column: “the earliest Olympic event”
  - hop between two cells: “Which event ... participate in ...”,
  - extract information from table: “In which year did the player ... ”.
- **Text reasoning step specifically includes**
  - select passages based on mentions, e.g. “the judoka bearer”,
  - extract a span from the passage as the answer.

# HybridQA using SUQL

- HybridQA is much more challenging than Yelp
- Each question comes with its own DB schema
- In-context learning: Zero-Shot
  - Schema of the table
  - No table-specific examples

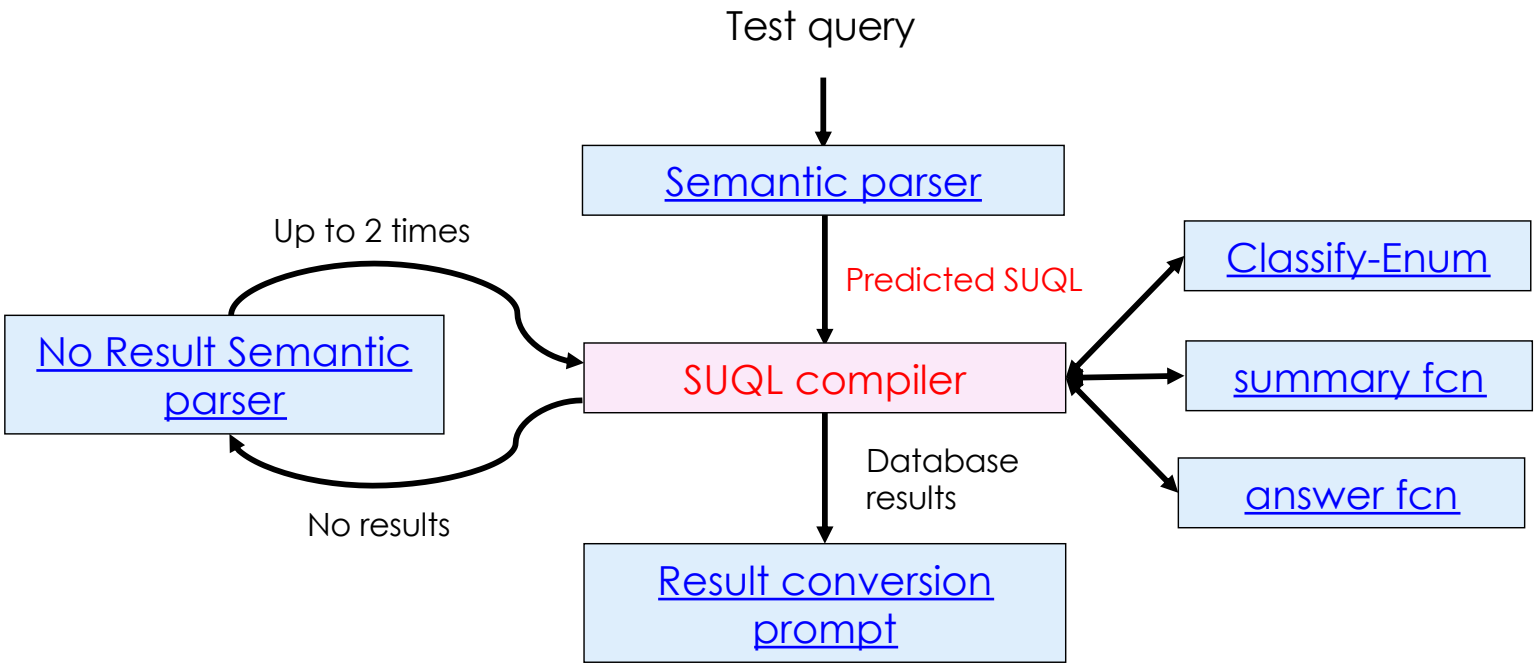
# Preliminary Assessment

Our HybridQA experiment suggests 2 improvements to SUQL

1. It is hard for the semantic parser to identify the right text column
   - Solution:
     Automatically find the answers in other columns
     if the original column fails

2. A clear feedback signal at run time: execution returns no answer
   - Solution: Try again

# SUQL-based agent on HybridQA
# A simple prompt-based system




Templates written in jinja syntax

# HybridQA Tables

| #Table | #Row/#Column | #Cell | #Links/Table |
|---|---|---|---|
| 13,000 | 15.7/4.4 | 70 | 44 |
| #Passage | #Words/Passage | #Ques | #Words/Ques |
| 293,269 | 103 | 69,611 | 18.9 |

- SUQL translates text tables into SQL table
- **Note the size of the tables**
  - Only 4.4 columns and 15.7 rows per table on average!
- SOTA $S^3$HQA works on supplying the whole table to LLMs!
  - Does **NOT** generalize to bigger tables
- Our solution scales to any table sizes

# Train – Dev – Test Split

| Split | Train | Dev | Test | Total |
|---|---|---|---|---|
| In-Passage | 35,215 | 2,025 | 20,45 | 39,285 (56.4%) |
| In-Table | 26,803 | 1,349 | 1,346 | 29,498 (42.3%) |
| Computed | 664 | 92 | 72 | 828 (1.1%) |
| Total | 62,682 | 3,466 | 3,463 | 69,611 |

Table 3: Data Split: In-Table means the answer comes from plain text in the table, and In-Passage means the answer comes from certain passage.

# Evaluation in HybridQA

Evaluation Metrics: Exact match (EM) and F1 on raw text

| Method | Model | Trained on (Size) | Dev | | Test | |
|---|---|---|---|---|---|---|
| | | | EM | F1 | EM | F1 |
| DocHopper (Sun et al., 2022) | ETC | HybridQA (62k) | 47.7 | 55.0 | 46.3 | 53.3 |
| HYBRIDER (Chen et al., 2020) | BERT | | 44.0 | 50.7 | 43.8 | 50.6 |
| MuGER[2] (Wang et al., 2022) | | | 57.1 | 67.3 | 56.3 | 66.2 |
| Mate (Eisenschlos et al., 2021) | | | 63.4 | 71.0 | 62.8 | 70.2 |
| DEHG (Feng et al., 2022) | | | 65.2 | **76.3** | 63.9 | 75.5 |
| MITQA (Kumar et al., 2023) | | | 65.5 | 72.7 | 64.3 | 71.9 |
| MAFiD (Lee et al., 2023b) | T5 | | 66.2 | 74.1 | 65.4 | 73.6 |
| $S^3$HQA (Lei et al., 2023b) | BERT/BART/DeBERTa | | **68.4** | 75.3 | **67.9** | **75.5** |
| LLaMA2 (7B) (Zhang et al., 2023a) | LLaMA2 (7B) | Zero-shot | 20.7 | - | - | - |
| TableLlama (Zhang et al., 2023a) | | TableInstruct (2.6M) | 27.6 | - | - | - |
| End-to-End QA w/ retriever (Shi et al., 2024) | GPT-4 | Zero-shot | 24.5† | 30.0† | - | - |
| HPROPRO (Shi et al., 2024) | | Few-shot | 48.0† | 54.6† | 48.7 | 57.7 |
| SUQL (**Ours**) | | | 59.3 | 68.3 | 59.0 | 68.4 |

Table 3: Performance of few-shot-based SUQL and related work on the HybridQA dataset. † denotes running on 200 sampled cases from the development set (Shi et al., 2024).

In-context learning SUQL outperforms all in-context learning methods

Within 8.9% EM and 7.1% F1 to the SOTA trained on 62K examples

# QUIZ

## WHAT SHOULD WE DO NEXT?

# Error Analysis

We analyzed 72 randomly sampled error cases

**Evaluation Issues (61% of errors)**

- **Format mismatches**: **37.5%**. Predicted "Johnson City, Tenessee" vs. "Johnson City".
- **The gold label is either wrong or incomplete**: **23.6%**
  - There is only 1 gold answer when there should be more!

**True Errors (39% of errors)**

- Semantic parsing errors: 22.2%
- Errors from LLM-based functions: 11.1%
- Type-related conversation errors: 5.6%

**A lot of format problems !** ← Handled by fine-tuning, but it is not intrinsically important

**True accuracy may reach 84.2%**

# Conclusion: SUQL

- **High-Level Query Language for any Domain of Hybrid Data**
  - Expressiveness: Unifies the hybrid data sources
    - Arbitrary composition of free-text and DB knowledge
- **Domain-agnostic SUQL-Based Genie Agent Framework**
  - Developer supplies: tables, free texts, examples (optional)
- **SUQL decomposes problem into:**
    - Zero-Shot semantic parser: IR + relational algebra
    - An optimizing compiler
- Evaluation
  - **Needed in real-life conversations (Yelp)**
  - **In-context learning with LLMs works well for natural queries**