CMPUT 692 – Topics in Data Management with LLMs
Assignment 1

Daniel Penner
1908155

September 29, 2025

All associated code for this assignment can be found on my Github.

1 Query Selection

1.1 Query List

Table 1: The selected subset of queries.

Query ID	Database	Difficulty
91	financial	Simple
96	financial	Simple
101	financial	Simple
108	financial	Simple
117	financial	Moderate
128	financial	Moderate
136	financial	Moderate
149	financial	Challenging
169	financial	Challenging
173	financial	Challenging
393	card_games	Simple
394	card_games	Simple
406	card_games	Simple
411	card_games	Simple
427	card_games	Moderate
432	card_games	Moderate
434	card_games	Moderate
477	card_games	Challenging
507	card_games	Challenging
513	card_games	Challenging
547	codebase_community	Simple
550	codebase_community	Simple
556	codebase_community	Simple
558	codebase_community	Simple

Table 1 (continued)

Query ID	Database	Difficulty
572	codebase_community	Moderate
578	codebase_community	Moderate
587	codebase_community	Moderate
598	codebase_community	Challenging
639	codebase_community	Challenging
701	codebase_community	Challenging
789	superhero	Simple
793	superhero	Simple
795	superhero	Simple
806	superhero	Simple
814	superhero	Moderate
817	superhero	Moderate
823	superhero	Moderate
829	superhero	Challenging
834	superhero	Challenging
835	superhero	Challenging
1470	debit_card_specializing	Simple
1477	debit_card_specializing	Simple
1476	debit_card_specializing	Challenging
1478	debit_card_specializing	Simple
1486	debit_card_specializing	Simple
1490	debit_card_specializing	Moderate
1501	debit_card_specializing	Moderate
1516	debit_card_specializing	Moderate
1526	debit_card_specializing	Challenging
1531	debit_card_specializing	Moderate

1.2 Justification

The 5 categories were randomly selected. For each category 10 queries were chosen: 4 simple, 3 moderate, 3 challenging (with the exception of ”debit card specializing” which only had 2 challenging).

1.3 Methodology

1.3.1 Model

The model used for this assignment was GPT-5-mini. A closed-weight model from OpenAI. I selected this model because of its lightweight, and thus resource efficient, nature, along with the fact that it is among the newest and therefore most minimally tested OpenAI models. All settings on the model were left default, partially because GPT-5-mini does not offer control over settings like temperature or Top-p, and partially to test the out of the box effectiveness of this model. At default settings the model has the `reasoning_effort` set to minimal and `verbosity` set to medium.

1.3.2 Code

The SQL predictions were generated by the `sql_generation.mjs` file at the root of my assignment repository. This file does the following:

1. Parses the subset of queries that I selected from the `input_data/queries.json` file.
2. Loops through each query and for each:
   (a) Finds the schema associated with the database that the query is used on.
   (b) Translates the schema into columns and rows.
   (c) Queries the model with a list of rules, the formatted schema, and the current query.
3. Prints the model's response to the `evaluation_repo/sql_result/predictions.json` file.

1.3.3 Query

The following data was fed to the model for each question:

```
Translate the natural language question into a simple, effective SQLite query.

RULES:
1. Use ONLY the provided table and column names.
2. Use ONLY names that appear in the schema description (no synonyms).
3. Do not invent new tables, columns, or values.
4. Output must be a single line with no explanations or non-SQL text.
5. Prefer the simplest correct form that exactly matches the question intent.

Schema:
Question:
```

Where ”Schema: ” is followed by a list of the schema elements in the form of

```
Tables: <table1>, <table2>, ...
Columns: <column1>, <column2>, ...
```

from the database schema associated with the question and ”Question” is followed by the provided natural language question.

2 Evaluation

2.0.1 Methodology

The official evaluation scripts in the `mini_dev` repository were tailored around use for a specific subset of 500 queries from the dev set. Because of this, I had to generate my own `mini_dev_sqlite.jsonl` and `mini_dev_sqlite_gold.sql` files using the correct SQL associated with each of the natural language questions I selected. They can be found in `evaluation_repo/evaluation/sqlite`. There should also be the complete `dev_databases` file there which cannot be committed to github due to it's sheer size. This has been locally added and placed in .gitignore on both of my personal devices used for this assignment. These changes were made along with additional logging used to identify issues during the evaluation step.

2.0.2 Results

Three metrics were used to evaluate the LLMs SQL queries:

1. Execution Accuracy (EX) - Whether the LLMs query has the same output as the gold query.
2. Reward Value Execution Score (R-VES) - Tests both the correctness and the efficiency relative to the gold query. R-VES was run with 10 iterations per CPU on 16 CPUs simultaneously. Each iteration had a time-out of 2 seconds.
3. Soft-F1 (EX) - Measures the similarity between the LLMs queries and the gold queries regardless of correctness.

Errors in SQL queries and failures to execute are simply treated as inaccuracies on the part of the LLM, and detract from both the EX and R-VES without halting the running of tests.

Table 2: Evaluation results across difficulty levels.

Metric	**Simple (%)**	**Moderate (%)**	**Challenging (%)**	**Total (%)**
EX	35.00	37.50	28.57	34.00
R-VES	17.42	17.08	21.43	18.43
Soft-F1	35.03	44.05	25.00	35.11

3 Analysis

3.1 Visual Trends

From simply inspecting the gold SQL file beside the LLM's predictions, it is apparent that the predicted queries are generally longer and more complex than the gold queries despite the instructions to the model being very clear about keeping answers as simple as possible. This could contribute to the R-VES being significantly lower than the EX and Soft-F1, as many of the queries are technically correct, but run slowly. This is likely caused by the default model verbosity being set to medium.

3.2 Difficulty Comparison

Strangely, difficulty had a more minimal impact on accuracy than expected. One would theorize that simpler questions would be easier for the model to solve, but the EX and Soft-F1 values are highest for moderate difficulties. The R-VES is even stranger, being highest for the challenging queries. This could imply that while the model's solutions vary heavily from the gold SQL, the optimal queries for the challenging questions are slower and thus the unique solutions concocted by the model have a reduced difference in speed compared to the simple queries where the gold is much quicker.

4 Comparison

4.1 Selected Models for Comparison

The models papers that were selected for comparison with the aforementioned results were the RSL-SQL technique using DeepSeek-v2 [1] and the SuperSQL model [2], placed 44th and 55th on the leaderboard respectively.

4.2 Reported Results

Both models published results for Execution Accuracy and Valid Efficiency scores using the official BIRD evaluation tool for the entire BIRD Dev dataset. These results are compared to those of the GPT-5-mini output in Table 3.

Model / Method	Simple	Moderate	Challenging	Total
Execution Accuracy (EX, %)				
GPT-5-mini (Selected Subset)	35.00	37.50	28.57	34.00
SuperSQL (BIRD dev)	66.90	46.50	43.80	58.50
RSL-SQL + DeepSeek (BIRD dev)	69.73	54.09	54.48	63.56
Valid Efficiency Score / R-VES (%)				
GPT-5-mini (Selected Subset)	17.42	17.08	21.43	18.43
SuperSQL (BIRD dev)	69.75	50.55	49.08	61.99
RSL-SQL + DeepSeek (BIRD dev)	–	–	–	67.68
Soft-F1 (%)				
GPT-5-mini (Selected Subset)	35.03	44.05	25.00	35.11

Table 3: Comparison of evaluation results with GPT-5-mini in this assignment on a 50 query subset [A] with RSL-SQL + Deepseek [1] and SuperSQL [2] on the BIRD development set.

4.3 Adjusted for 50 Query Subset

Both the RSL-SQL + DeepSeek-v2 [1] and SuperSQL [2] papers published their generated sets of SQL queries for the BIRD Dev dataset. Using these, I created subsets of their SQL output for the same queries I ran against GPT-5-mini. These were evaluated using the same tools used to evaluate the GPT-5-mini output. Results compared to my own can be found in Table 4.

Model / Method	Simple	Moderate	Challenging	Total
Execution Accuracy (EX, %)				
GPT-5-mini	35.00	37.50	28.57	34.00
RSL-SQL + DeepSeek	75.00	62.50	64.29	68.00
SuperSQL	60.00	56.25	50.00	56.00
Valid Efficiency Score / R-VES (%)				
GPT-5-mini	17.42	17.08	21.43	18.43
RSL-SQL + DeepSeek	38.48	29.58	36.24	35.00
SuperSQL	24.82	29.58	21.43	25.39
Soft-F1 (%)				
GPT-5-mini	35.03	44.05	25.00	35.11
RSL-SQL + DeepSeek	75.00	68.67	60.71	68.97
SuperSQL	60.00	62.21	42.86	55.91

Table 4: Comparison of evaluation results with GPT-5-mini in this assignment with RSL-SQL + Deepseek [1] and SuperSQL [2] all using the aforementioned 50 query subset [A].

4.4 Strengths and Weaknesses of this Approach

The SuperSQL model and combination of the RSL-SQL technique with the Deepseek both achieved scores for both accuracy and efficiency significantly above what GPT-5-mini accomplished using the techniques presented in this paper. Both papers for these approaches involved significant training, while this attempt only involved prompting and the full schema being included with each question. Additionally, RSL-SQL involved schema pruning, SuperSQL utilized post-processing, and both papers used schema linking to further optimize the model's performance. All of these techniques are very effective ways of improving the model's understanding of the database associated with a given query.

Both of these papers were assessed against the entire dev dataset which includes over 1500 queries, whereas this report covered only 50. This smaller sample may have affected the average difficulty of queries either positively or negatively for this assignment.

This report involves the analysis of the responses' Soft-F1 scores, a unique metric not employed by this previous research. This provides valuable insight outside of the scope of what is covered in these papers.

5 Resources

- OpenAI documentation was used to aid in the implementation of the GPT-5-mini API into the code [3].
- All evaluation code was either taken directly or modified slightly after being taken directly from the `bird_bench/mini_dev` repository on GitHub [4].
- ChatGPT was used to assist in error identification while programming, altering evaluation code, translating JSON files and raw data into lists and tables, and assisting with coding practices for this report [5].

A Generated SQL Queries

Below is the full set of generated SQL queries for the selected natural language questions. Each query is referenced by its `question_id`.

```
91: SELECT CASE WHEN AVG(district.A6)>AVG(district.A7) THEN '1995'
               WHEN AVG(district.A6)<AVG(district.A7) THEN '1996'
               ELSE 'tie' END AS higher_year,
           AVG(district.A6) AS avg_1995,
           AVG(district.A7) AS avg_1996
    FROM district;
```

```
96: SELECT COUNT(DISTINCT disp.client_id)
    FROM disp
    JOIN account ON disp.account_id = account.account_id
    WHERE UPPER(disp.type) = 'OWNER'
      AND UPPER(account.frequency) LIKE '%WEEK%';
```

```
101: SELECT DISTINCT account.account_id
     FROM account
     JOIN trans ON account.account_id = trans.account_id
     WHERE substr(trans.date,1,4)='1995'
       AND trans.date = (SELECT MIN(date)
                         FROM trans
                         WHERE substr(date,1,4)='1995');
```

```
108: SELECT t.amount
     FROM trans t
     JOIN account a ON t.account_id=a.account_id
     JOIN loan l ON l.account_id=a.account_id
     WHERE l.amount=(SELECT MAX(amount) FROM loan)
       AND t.date>=a.date
     ORDER BY t.date
     LIMIT 1;

117: SELECT COALESCE(
       100.0 * SUM(CASE WHEN loan.status = 'closed' THEN loan.amount ELSE 0 END)
       / NULLIF(SUM(loan.amount),0), 0)
     FROM loan;

128: SELECT district.A2, COUNT(DISTINCT client.client_id) AS cnt
     FROM client
     JOIN disp ON client.client_id=disp.client_id
     JOIN account ON disp.account_id=account.account_id
     JOIN district ON account.district_id=district.district_id
     WHERE client.gender='F' AND disp.type='OWNER'
     GROUP BY district.A2
     ORDER BY cnt DESC
     LIMIT 9;

136: SELECT loan.account_id, COUNT(*)
     FROM loan
     JOIN account ON loan.account_id=account.account_id
     WHERE loan.date BETWEEN '1995-01-01' AND '1997-12-31'
       AND loan.amount>=250000
       AND account.frequency='M'
       AND loan.status='approved'
     GROUP BY loan.account_id;

149: SELECT DISTINCT account.frequency
     FROM account
     JOIN district ON account.district_id = district.district_id
     WHERE district.A11>8000 AND district.A11<=9000
       AND account.account_id NOT IN (
           SELECT account_id FROM loan
       );
```

169:
```
SELECT CASE WHEN t.s1996=0 THEN NULL
            ELSE (t.s1997 - t.s1996)*1.0 / t.s1996 END AS growth
FROM (
  SELECT SUM(CASE WHEN strftime('%Y',loan.date)='1996' THEN loan.amount ELSE 0 END) AS s1996,
         SUM(CASE WHEN strftime('%Y',loan.date)='1997' THEN loan.amount ELSE 0 END) AS s1997
  FROM loan
  JOIN disp ON loan.account_id=disp.account_id
  JOIN client ON disp.client_id=client.client_id
  WHERE client.gender='M'
) t;
```

173:
```
SELECT account.frequency,
       (SELECT trans.k_symbol
        FROM trans
        WHERE trans.account_id=3 AND trans.type='Debit'
        GROUP BY trans.k_symbol
        HAVING SUM(trans.amount)=3539)
FROM account
WHERE account.account_id=3;
```

393:
```
SELECT COUNT(*)
FROM cards
WHERE artist = 'John Avon' AND hasFoil = 0;
```

394:
```
SELECT COUNT(*)
FROM cards
WHERE borderColor = 'white' AND CAST(power AS INTEGER) >= 5;
```

406:
```
SELECT DISTINCT cards.id
FROM cards
JOIN legalities ON cards.uuid = legalities.uuid
WHERE cards.type LIKE '%Creature%';
```

411:
```
SELECT cards.artist
FROM cards
WHERE cards.text = 'Das perfekte Gegenmittel zu einer dichten Formation';
```

427:
```
SELECT DISTINCT language
FROM set_translations
WHERE setCode = 'ARC';
```

432:
```
SELECT sets.name
FROM sets
JOIN set_translations ON set_translations.setCode = sets.code
WHERE set_translations.language = 'Russian'
ORDER BY sets.totalSetSize DESC
LIMIT 1;
```

434:
```
SELECT COUNT(*)
FROM sets s
WHERE s.isOnlineOnly = 0
  AND EXISTS (
    SELECT 1 FROM set_translations st
    WHERE st.setCode = s.code AND st.language = 'Japanese'
  )
  AND NOT EXISTS (
    SELECT 1 FROM set_translations st2
    WHERE st2.setCode = s.code AND st2.language != 'Japanese'
  );
```

477:
```
SELECT DISTINCT cards.artist
FROM cards
JOIN sets ON cards.setCode = sets.code
WHERE sets.name = 'Coldsnap'
  AND cards.artist IN ('Jeremy■Jarvis','Aaron■Miller','Chippy');
```

507:
```
SELECT 100.0 * SUM(isOnlineOnly) / COUNT(*) AS percentage
FROM (
  SELECT DISTINCT sets.code, sets.isOnlineOnly
  FROM sets
  JOIN set_translations ON sets.code = set_translations.setCode
  WHERE set_translations.language = 'Portuguese■(Brazil)'
);
```

513:
```
SELECT id
FROM sets
WHERE type = 'Commander'
ORDER BY totalSetSize DESC
LIMIT 1;
```

547:
```
SELECT COUNT(*)
FROM posts
JOIN users ON posts.OwnerUserId = users.Id
WHERE users.Age >= 60 AND posts.Score > 19;
```

550:
```
SELECT posts.Body
FROM posts
JOIN tags ON tags.ExcerptPostId = posts.Id
ORDER BY tags.Count DESC
LIMIT 1;
```

556:
```
SELECT AVG(badge_count)
FROM (
  SELECT users.Id, COUNT(badges.Id) AS badge_count
  FROM users
```

```
    LEFT JOIN badges ON users.Id = badges.UserId
    WHERE users.Views > 200
    GROUP BY users.Id
  );
```

558:
```
SELECT COUNT(*)
FROM votes
JOIN posts ON votes.PostId = posts.Id
WHERE posts.OwnerUserId = 58
  AND date(votes.CreationDate) = '2010-07-19';
```

572:
```
SELECT posts.ViewCount
FROM posts
WHERE posts.Title = 'Integration■of■Weka■and/or■RapidMiner■into■Informatica■PowerCenter/Developer';
```

578:
```
SELECT users.DisplayName, users.Reputation
FROM posts
JOIN users ON posts.OwnerUserId = users.Id
WHERE posts.Title = 'Understanding■what■Dassault■iSight■is■doing?';
```

587:
```
SELECT (SELECT AVG(ViewCount)
        FROM posts
        WHERE Tags LIKE '%<humor>%') AS AvgViewCount,
       posts.Title, comments.Text
FROM posts
LEFT JOIN comments ON comments.PostId = posts.Id
WHERE posts.Tags LIKE '%<humor>%';
```

598:
```
SELECT CASE WHEN c0=0 THEN NULL
            ELSE (c1 - c0) * 100.0 / c0 END AS percentage_difference
FROM (
  SELECT SUM(CASE WHEN Name='student' AND strftime('%Y',Date)='2010' THEN 1 ELSE 0 END) AS c0,
         SUM(CASE WHEN Name='student' AND strftime('%Y',Date)='2011' THEN 1 ELSE 0 END) AS c1
  FROM badges
);
```

639:
```
SELECT 100.0 * SUM(CASE WHEN posts.Tags LIKE '%<r>%' THEN 1 ELSE 0 END)
       / COUNT(*) AS percentage
FROM posts
WHERE posts.OwnerDisplayName = 'Community';
```

701:
```
SELECT 100.0 * SUM(CASE WHEN posts.Score > 50 THEN 1 ELSE 0 END)
       / NULLIF(COUNT(*),0) AS percentage
FROM posts
WHERE posts.OwnerUserId = (
  SELECT users.Id
  FROM users
  ORDER BY users.Reputation DESC
  LIMIT 1
);
```

```
789: SELECT AVG(superhero.weight_kg)
     FROM superhero
     JOIN race ON superhero.race_id = race.id
     WHERE race.race = 'Alien';

793: SELECT COUNT(*)
     FROM superhero
     JOIN race ON superhero.race_id = race.id
     JOIN gender ON superhero.gender_id = gender.id
     WHERE race.race = 'god/eternal' AND gender.gender = 'male';

795: SELECT COUNT(*)
     FROM superhero
     JOIN alignment ON superhero.alignment_id = alignment.id
     WHERE lower(alignment.alignment) = 'neutral';

806: SELECT colour.colour
     FROM superhero
     JOIN colour ON superhero.eye_colour_id = colour.id
     WHERE superhero.full_name = 'Karen■Beecher-Duncan';

814: SELECT DISTINCT colour.colour
     FROM superhero
     JOIN hero_attribute ON superhero.id = hero_attribute.hero_id
     JOIN colour ON superhero.skin_colour_id = colour.id
     WHERE hero_attribute.attribute_value = 100;

817: SELECT race.race
     FROM superhero
     JOIN gender ON superhero.gender_id = gender.id
     JOIN colour ON superhero.hair_colour_id = colour.id
     JOIN race ON superhero.race_id = race.id
     WHERE gender.gender = 'Male' AND colour.colour = 'Blue';

823: SELECT COUNT(*)
     FROM superhero
     JOIN gender ON superhero.gender_id = gender.id
     JOIN publisher ON superhero.publisher_id = publisher.id
     WHERE gender.gender = 'Female' AND publisher.publisher_name = 'Marvel■Comics';

829: SELECT CASE WHEN dc_count>marvel_count THEN 'DC'
                 WHEN dc_count<marvel_count THEN 'Marvel■Comics'
                 ELSE 'Tie' END AS publisher,
            abs(dc_count-marvel_count) AS difference
     FROM (
       SELECT (SELECT COUNT(*)
            FROM superhero s
            JOIN publisher p ON s.publisher_id=p.id
            WHERE p.publisher_name='DC') AS dc_count,
           (SELECT COUNT(*)
            FROM superhero s2
            JOIN publisher p2 ON s2.publisher_id=p2.id
            WHERE p2.publisher_name='Marvel■Comics') AS marvel_count
);
```

834:
```
SELECT 100.0 * SUM(CASE WHEN gender.gender = 'Female' THEN 1 ELSE 0 END)
       / COUNT(*)
FROM superhero
JOIN publisher ON superhero.publisher_id = publisher.id
LEFT JOIN gender ON superhero.gender_id = gender.id
WHERE publisher.publisher_name = 'George■Lucas';
```

835:
```
SELECT AVG(CASE WHEN alignment.alignment='good' THEN 1.0 ELSE 0 END)*100 AS percentage
FROM superhero
JOIN publisher ON superhero.publisher_id = publisher.id
LEFT JOIN alignment ON superhero.alignment_id = alignment.id
WHERE publisher.publisher_name='Marvel■Comics';
```

1470:
```
SELECT COUNT(DISTINCT gasstations.GasStationID)
FROM gasstations
JOIN transactions_1k ON gasstations.GasStationID = transactions_1k.GasStationID
JOIN products ON transactions_1k.ProductID = products.ProductID
WHERE gasstations.Country = 'CZE' AND products.Description = 'Premium';
```

1476:
```
SELECT SUM(CASE WHEN customers.Currency='CZK' THEN yearmonth.Consumption ELSE 0 END)
     - SUM(CASE WHEN customers.Currency='EUR' THEN yearmonth.Consumption ELSE 0 END) AS difference
FROM yearmonth
JOIN customers ON yearmonth.CustomerID=customers.CustomerID
WHERE yearmonth.Date LIKE '2012%';
```

1477:
```
SELECT strftime('%Y', transactions_1k.Date) AS Year
FROM transactions_1k
JOIN customers ON transactions_1k.CustomerID = customers.CustomerID
WHERE customers.Currency = 'EUR'
GROUP BY Year
ORDER BY SUM(transactions_1k.Amount) DESC
LIMIT 1;
```

1478:
```
SELECT customers.Segment
FROM yearmonth
JOIN customers ON yearmonth.CustomerID = customers.CustomerID
GROUP BY customers.Segment
ORDER BY SUM(yearmonth.Consumption) ASC
LIMIT 1;
```

1486:
```
SELECT CASE WHEN SUM(CASE WHEN Currency='Czech■koruna' THEN 1 ELSE 0 END) >
                 SUM(CASE WHEN Currency='euro' THEN 1 ELSE 0 END)
              THEN 'true' ELSE 'false' END AS more_czech,
       SUM(CASE WHEN Currency='Czech■koruna' THEN 1 ELSE 0 END) -
       SUM(CASE WHEN Currency='euro' THEN 1 ELSE 0 END) AS difference
FROM customers
WHERE Segment='SME';
```

1490:
```
SELECT (SELECT COUNT(DISTINCT yearmonth.CustomerID)
        FROM yearmonth
        JOIN customers ON yearmonth.CustomerID=customers.CustomerID
        WHERE customers.Segment='LAM' AND yearmonth.Consumption>46.73)*100.0
        /(SELECT COUNT(DISTINCT CustomerID)
          FROM customers
          WHERE Segment='LAM');
```

1501:
```
SELECT DISTINCT gasstations.Country
FROM gasstations
JOIN transactions_1k ON gasstations.GasStationID = transactions_1k.GasStationID
WHERE transactions_1k.Date LIKE '2013-06%';
```

1516:
```
SELECT COUNT(*)
FROM transactions_1k
JOIN customers ON transactions_1k.CustomerID = customers.CustomerID
WHERE customers.Currency = 'CZK'
  AND transactions_1k.Date = '2012/8/26'
  AND transactions_1k.Time < '12:00:00';
```

1526:
```
SELECT (t.m2012 - t.m2013) / t.m2012 AS decrease_rate
FROM (
  SELECT MAX(CASE WHEN Date='2012' THEN Consumption END) AS m2012,
         MAX(CASE WHEN Date='2013' THEN Consumption END) AS m2013
  FROM yearmonth
  WHERE CustomerID = (
    SELECT CustomerID
    FROM transactions_1k
    WHERE Date='2012/8/25' AND Amount = 634.8
    LIMIT 1
  )
) AS t;
```

1531:
```
SELECT customers.CustomerID,
       SUM(transactions_1k.Price) AS total_spent,
       SUM(transactions_1k.Price)/SUM(transactions_1k.Amount) AS avg_price_per_item,
       customers.Currency
FROM transactions_1k
JOIN customers ON transactions_1k.CustomerID=customers.CustomerID
GROUP BY customers.CustomerID, customers.Currency
ORDER BY total_spent DESC
LIMIT 1;
```

References

[1] Z. Cao, Y. Zheng, Z. Fan, X. Zhang, W. Chen, and X. Bai, “RSL-SQL: Robust Schema Linking in Text-to-SQL Generation,” *arXiv preprint arXiv:2411.00073*, 2024.

[2] Boyan Li, Yuyu Luo, Chengliang Chai, Guoliang Li, Nan Tang. The Dawn of Natural Language to SQL: Are We Fully Ready?. PVLDB, 17(11): 3318 - 3331, 2024. doi:10.14778/3681954.3682003

[3] OpenAI API Documentation. Retrieved Sept. 2025 from https://platform.openai.com/docs/quickstart

[4] Bird-Bench. Bird-bench/mini_dev. Retrieved Sept. 2025 from https://github.com/bird-bench/mini_dev

[5] OpenAI, ChatGPT 5 *https://chat.openai.com/chat, 2025*