Natural Language Interfaces for Database Querying

Dr. Pınar Karagöz

METU, Computer Eng. Dept

ODTÜ METU

EURO² EURO 4SEE

Machine Interface: How do you talk to a Coffee Machine?

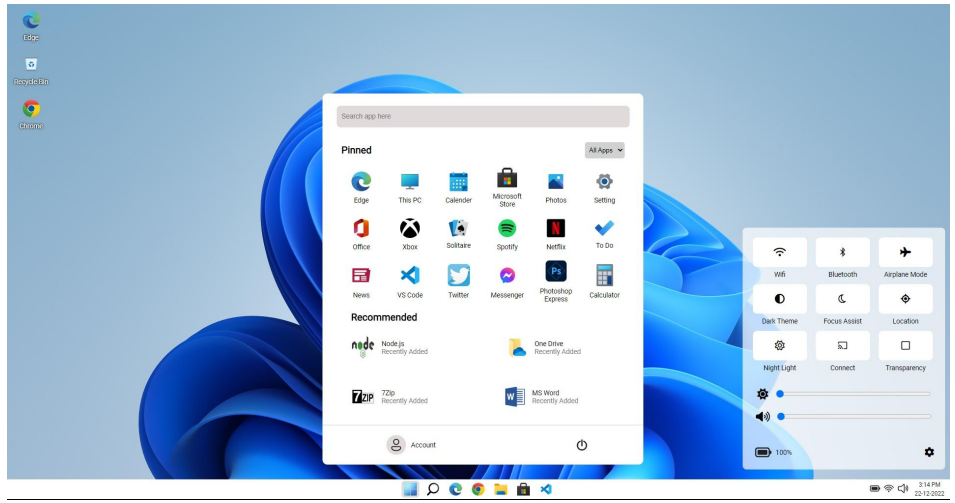
Machine Interface: How do you talk to a Coffee Machine?

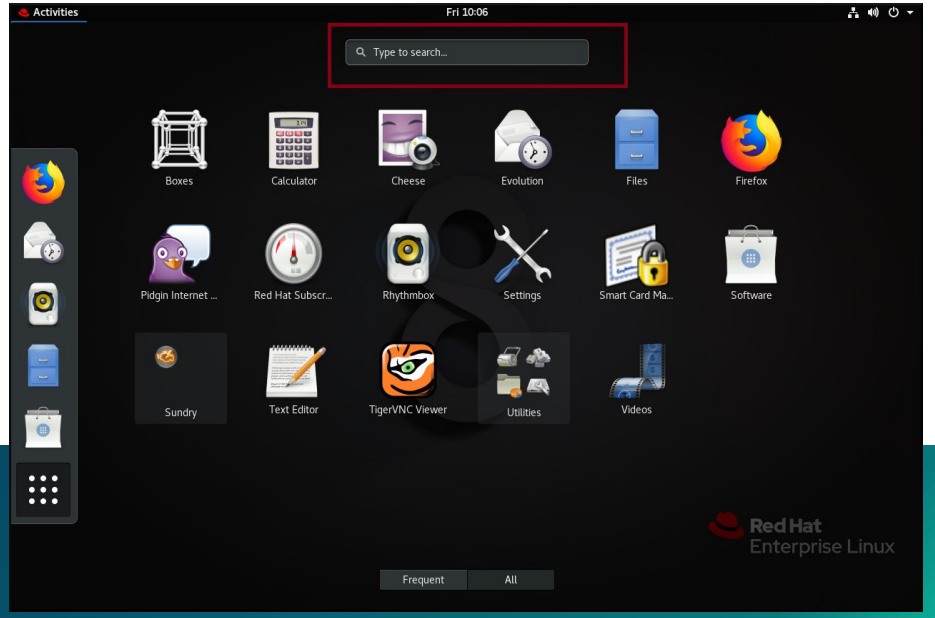
Source: https://dailycoffeenews.com/2016/05/31/antiquated-complicated-steampunk-coffee-machine-voted-most-innovative-of-london-coffee-festival/

Machine Interface: How do you talk to a Coffee Machine?

User Interface: the space where interactions between humans and machines occur. The goal of this interaction is to allow effective operation and control of the machine from the human end, while the machine simultaneously feeds back information that aids the operators' decision-making process (source: Wikipedia)

Computer Interface: How do you talk to a Computer





How about Natural Language Interface

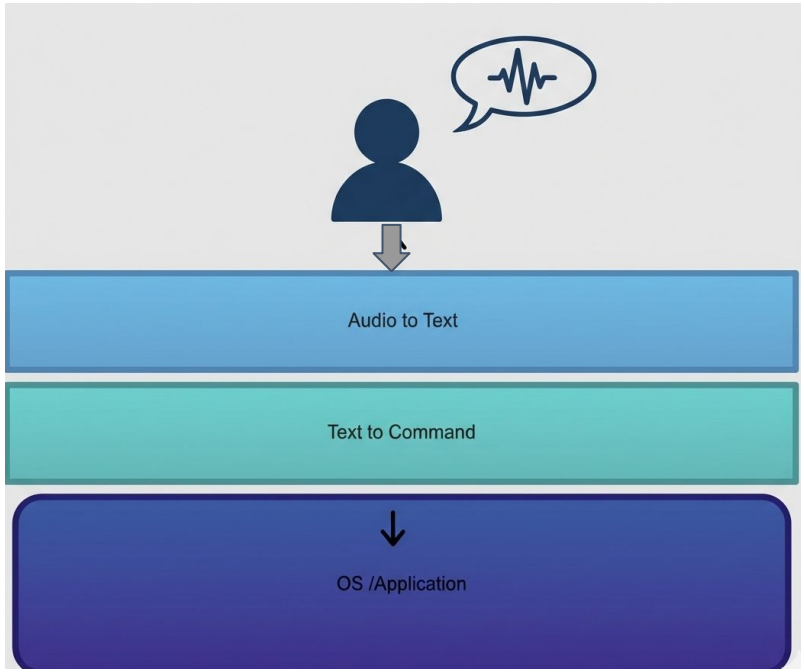
Communicating with a computer using natural language

How about Natural Language Interface

Communicating with a computer using natural language
E.g. Siri, Alexa, Google Assistant, Cortana/Copilot

Natural Language Interface

Communicating with a computer using natural language

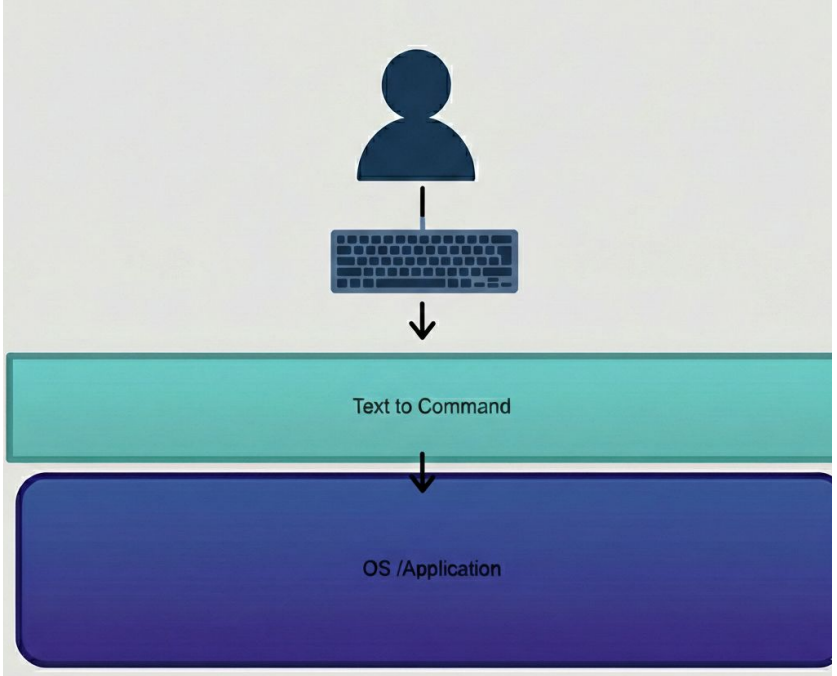


Audio-to-text / Speech-to-text:

Many pre-trained models.
One example:

Zhang, Y., Han, W., Qin, J., Wang, Y., Bapna, A., Chen, Z., ... & Wu, Y. (2023). Google usm: Scaling automatic speech recognition beyond 100 languages. *arXiv preprint arXiv:2303.01037*.

Natural Language Interface



Text-to-Command:

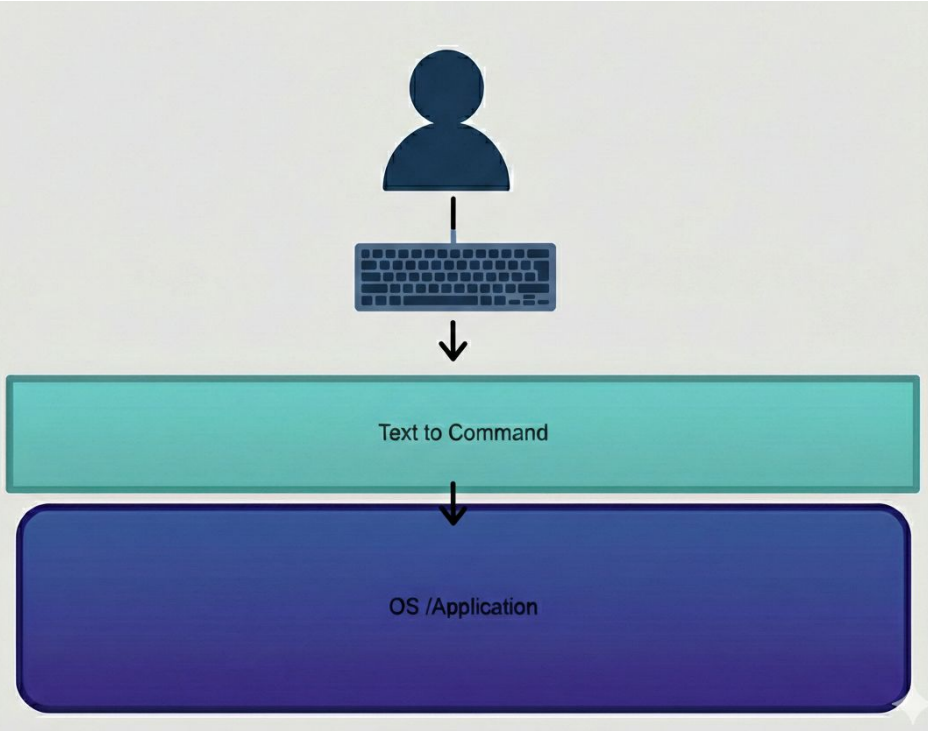
Traditionally Natural Language Processing + Machine Learning models have been used

NLI: Use of NLP

Natural Language Processing (NLP) Pipeline

- Basically Vector Space Model is used
- Needs preprocessing including stop word removal, stemming, dimension reduction (such as Latent Semantic Indexing)
- Term weighting (TF-IDF)
- Other tasks including
  - Named Entity Recognition
  - Part-of-Speech Tagging
  - Intent mining, …

NLI: Use of LLMs



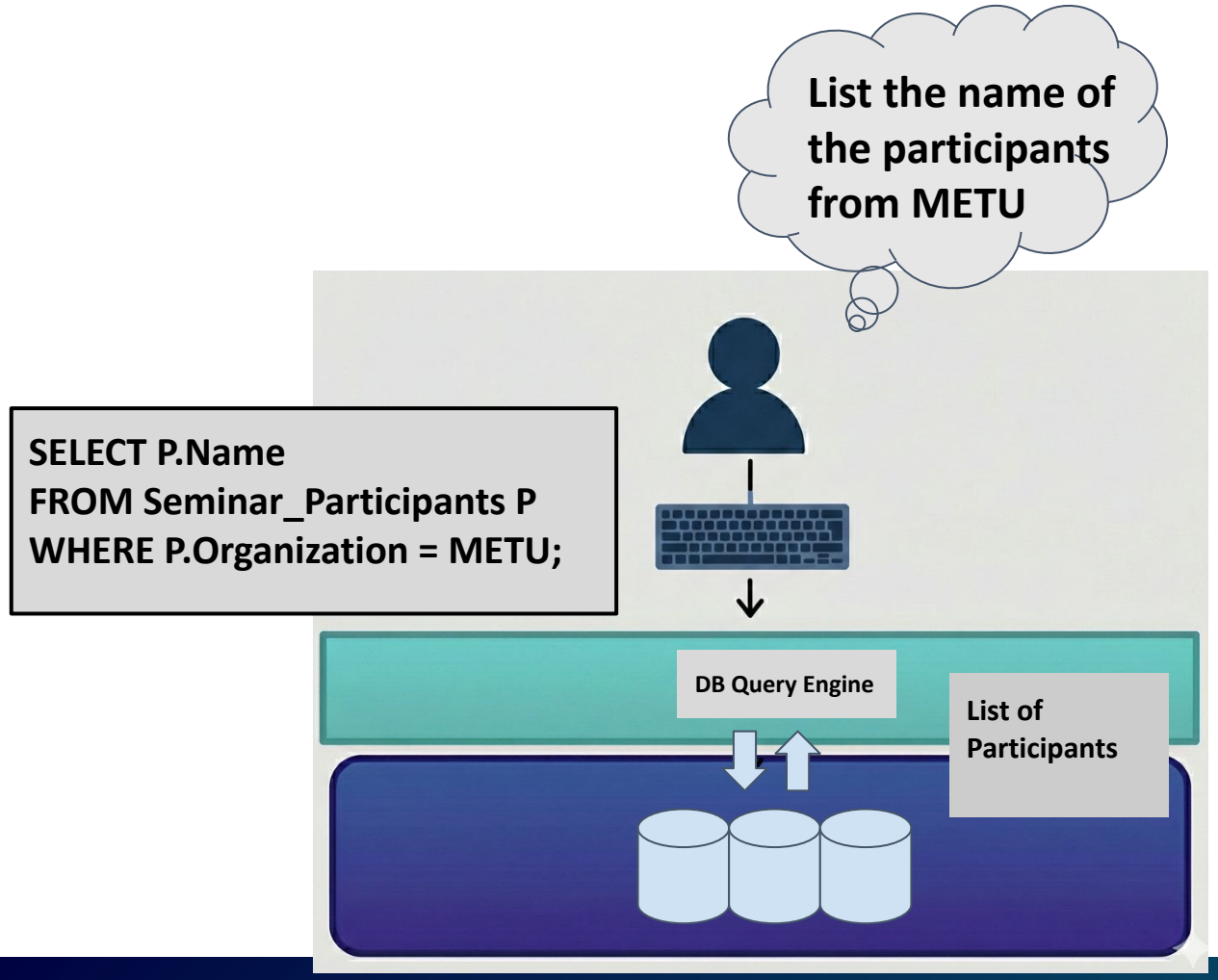
Text-to-Command:

Recent approaches are based on the use of Large Language Models (LLMs)

With Pre-trained/fine tuned/in-context learning mode for the target task

How about Natural Language Interface for Database Querying?

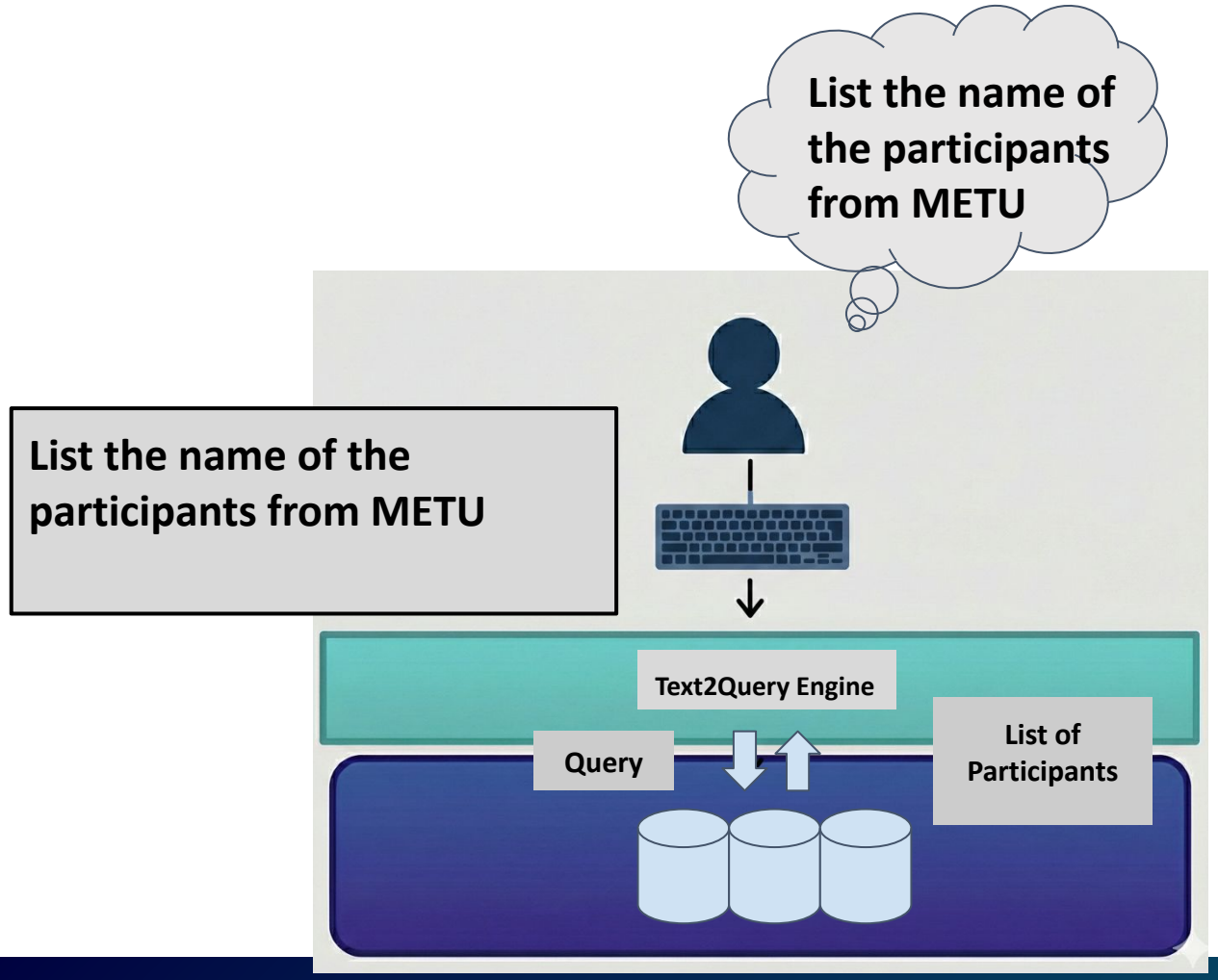
NLI: Text-to-Query



Classical Database Querying

- Write your query directly in DB language

NLI: Text-to-Query



Text-to-Query:

- Translate the natural query to database query
- Run the query against the database
- Show the result to the user

NLI: Text-to-Query

Why we would need it?

Extreme amount of data collected in variety of domains

But needs db query language knowledge for effective retrieval

→ **Lowering the data access barrier**

A Quick Look at Databases

SQL - Relational DBs

- Tabular structure
- Strict schema
- JOINs
- Standard query language - SQL

NoSQL

- Schema-less, flexible
- Nested structure
- Umbrella concept - Document DB, GraphDB, VectorDB
- No Standard Query Language

The Evolution of Text-to-SQL

Era 1: Rule-Based & Statistical (Pre-2017): Hand-crafted grammars and mapping rules.

Era 2: Neural Translation (2017–2022): Seq2Seq models, Encoder-Decoder architectures (e.g., RAT-SQL).

Era 3: LLM Zero/Few-Shot (2023–2024): GPT-4 and Claude 3 as translators using sophisticated prompting (DIN-SQL).

Era 4: Agentic Text-to-SQL (2025–Present): Multi-agent systems with "Self-Correction," "Semantic Memory," and "Test-Time Scaling."

Core Challenges in Modern T2Q

Schema Linking: Mapping NL entities to thousands of possible columns in enterprise DBs.

Domain Specificity: Understanding acronyms and business logic (e.g., "Churn" ≠ simple deletion).

Nested Complexity: Handling multi-level JOINs, EXISTS clauses, etc.

Hallucination: Generating columns or tables that do not exist in the schema.

Current SotA Paradigms

Agentic RAG (Retrieval-Augmented Generation) pattern:

1. **Contextual Retrieval:** Searching for relevant schema metadata and value descriptions.
2. **Modular Decomposition:** Breaking the query into sub-problems (Schema Pruning → Logic Planning → SQL Synthesis).
3. **Iterative Refinement:** Executing SQL in a sandbox and using error logs for self-repair.

Top Performing Models

- Agentar-Scale-SQL: Utilizes "Test-Time Scaling" to explore multiple reasoning paths before finalizing a query.

Wang, Y., Chen, H., Zhao, L., & Miller, K. (2026). **Agentar-Scale-SQL: Bridging the Reasoning Gap through Test-Time Scaling**. *arXiv preprint arXiv:2602.04102*.

- XiYan-SQL: A multi-generator ensemble framework that integrates domain-specific fine-tuning.

Liu, Z., Hu, J., Ji, L., Fu, J., Chen, B., Li, X., ... & Yan, N. (2024). **XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL**. *arXiv preprint (Accepted in EMNLP 2025)*

Top Performing Models

- QUVI-3 (Gemini 3 Pro based): High-reasoning agent optimized for very long context (Spider 2.0 benchmarks).

  Zhang, L., Chen, Y., & Siddhartha, R. (2025). **QUVI-3: High-Efficiency Agentic Text-to-SQL via Quantized Verification and Reward-Guided Search**. *Proceedings of the 2026 International Conference on Learning Representations (ICLR)*

- TCDataAgent-SQL: Focuses on contextual scaling engines for massive enterprise schemas.

  Shao, D., Wang, H., & Liu, Z. (2025). **TCDataAgent: Scalable and Data-Aware Agentic Workflows for Large-Scale Text-to-SQL**. *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (ACL)*, Vol. 1, pages 412–429.

Agentar-Scale-SQL

Instead of a single "shot," the model treats SQL generation as a search problem.

Workflow:

- Generates N candidate queries.
- Uses a Verifier Agent to execute candidates against a sample database.
- Filters by execution result and semantic similarity to the original intent.

XiYan-SQL (Ensemble approach)

Strategy: No single prompt fits all. XiYan-SQL uses a Selector to route tasks to specialized generators.

Multi-Generator Ensemble:

- Generator A: Optimized for complex JOINs.
- Generator B: Optimized for aggregation (GROUP BY, HAVING).
- Generator C: Optimized for dialect-specific syntax (e.g., Snowflake vs. BigQuery).

Integration: Merges results using a consensus-based voting mechanism.

"Agentic RAG" for Schema Pruning

For databases with >1,000 columns, the prompt becomes too large (context window overflow).

Schema Pruning Agent: Uses vector search (Embedding-based) or LLM-based filtering to select only the top 5–10 relevant tables.

Value Retrieval: Retrieves distinct values from columns to resolve ambiguity.

Execution-Guided Correction (Self-Repair)

If the generated SQL fails, the agent doesn't stop:

Error Analysis: Captures DBMS error messages (e.g., "Table not found," "Division by zero").

Feedback Loop: Feeds the error back into the LLM with the prompt: "Your previous SQL failed with [ERROR]. Fix the logic and ensure the column names match the schema provided."

Emerging Trend: Semantic Memory

Agent Semantic Memory (AgentSM): Agents store "trajectories" of successful queries.

Reuse: If a user asks a similar question, the agent retrieves the previous reasoning path, reducing token usage by 25-35% and improving accuracy.

Relevant Datasets - Spider 1.0 & WikiSQL

Spider 1.0 (2018): The foundational benchmark. Cross-domain, complex SQL.

Constraint: Small databases, mostly "clean" schemas.

WikiSQL (2017): Massive scale but very simple queries.

Status: Largely considered "solved" by modern LLMs.

The BIRD Benchmark (Big Bench for Large-scale Database)

Complexity: Higher than Spider. Includes large-scale databases across 37+ professional domains (e.g., finance, sports, blockchain).

Key Feature: Requires the model to use "External Knowledge" (e.g., a PDF of business rules) to understand the data.

Leaderboard Status: Currently the primary benchmark for industrial Text-to-SQL agents.

The BIRD Benchmark (Big Bench for Large-scale Database)

- **Natural Language Question:** "List the IDs of the accounts that have the highest frequency of insurance payments."
- **External Knowledge (Evidence):** "Frequency of insurance payments refers to the frequency column in the account table where the value is 'POPLATEK MESICNE' (monthly issuance)."
- **Database Context:** The system must know to join the account table with the disposition and trans tables to count specific transaction types.

```
SELECT T1.account_id
FROM account AS
T1 JOIN trans AS T2 ON
T1.account_id = T2.account_id
WHERE T1.frequency = 'POPLATEK MESICNE'
GROUP BY T1.account_id
ORDER BY COUNT(T2.trans_id) DESC
LIMIT 1;
```

The BIRD Benchmark (Big Bench for Large-scale Database)

Key BIRD-SQL Challenges

- **Value Mapping:** Translating "insurance payments" to a specific database string like 'POPLATEK MESICNE'.
- **Reasoning Efficiency:** BIRD-SQL evaluates not just if the query is correct, but if it is **efficient** enough to run on large-scale datasets without timing out.
- **Ambiguity:** Handling questions where the specific column name is not mentioned, requiring the model to infer the link from the "Evidence" provided in the benchmark.

Natural Language Query for SQL

BIRD-SQL

A Big Bench for Large-Scale Database Grounded Text-to-SQLs

About BIRD

BIRD (**BI**g Bench for La**R**ge-scale **D**atabase Grounded Text-to-SQL Evaluation) represents a pioneering, cross-domain dataset that examines the impact of extensive database contents on text-to-SQL parsing. BIRD contains over **12,751** unique question-SQL pairs, **95** big databases with a total size of **33.4 GB**. It also covers more than **37** professional domains, such as blockchain, hockey, healthcare and education, etc.

Paper

Code

Mini-Dev (500)

BIRD-CRITIC 1.0 (SQL)

LiveSQLBench!

BIRD-Interact

Train Set | 🔥 Dev Set

News

Overall Leaderboard | Single-Model Leaderboard

Leaderboard - Execution Accuracy (EX)

	Model	Code	Size	Oracle Knowledge	Dev (%)	Test (%)
	Human Performance *Data Engineers + DB Students*			✓		**92.96**
🏆1 Dec 16, 2025	AskData + GPT-4o *AT&T CDO - DSAIR* [Shkapenyuk et al. '25]		UNK	✓	77.64	**81.95**
🥈2 Sep 25, 2025	Agentar-Scale-SQL *Ant Group* [Pengfei Wang et al. '25]	[link]	UNK	✓	74.90	81.67
🥉3 July 14, 2025	LongData-SQL *LongShine AI Research*		UNK	✓	74.32	77.53
4 Jan 02, 2026	Zhiwen-Lingsi-Agent *China Telecom, TeleAI*		UNK	✓	73.53	76.63
5 Jan 26, 2026	DeepEye-SQL *Anonymous*		UNK	✓	73.53	76.58
6 Dec 4, 2025	Q-SQL *AWS-Quick Science* [Ravi Shankar et al.'25]		30B-3B-MoE	✓	72.99	76.47
	MIC^2-SQL					

Spider 2.0

Released in late 2024, it became the gold standard in 2025.

Real-world Environment: Uses actual BigQuery, Snowflake, and ClickHouse databases.

Scale: Schemas often exceed 1,000 columns.

Workflow-oriented: Requires generating multiple queries to solve a single business problem.

Spider 2.0

Example: Crypto Transaction Analysis

Database: bigquery-public-data.crypto_ethereum (A real-world cloud dataset with TBs of data).

Natural Language Task: *"Find the top 5 Ethereum addresses that have received the most total value from transactions in the last 30 days, but only include addresses that have also initiated at least one contract internal transaction during that same period. Show the address and the total value received."*

Challenges:

- **Massive Context:** The agent must browse a schema with dozens of tables, each having complex partitioning and clustering (e.g., transactions, traces, contracts).
- **Dialect Specifics:** Requires BigQuery-specific syntax for handling TIMESTAMP and NUMERIC types (high precision).
- **Multi-Step Reasoning:** It requires a join between transaction data (to find "value received") and internal trace data (to verify "contract interactions").

Spider 2.0

Example: Crypto Transaction Analysis

Database:
bigquery-public-data.crypto_ethereum (A real-world cloud dataset with TBs of data).
Natural Language Task: *"Find the top 5 Ethereum addresses that have received the most total value from transactions in the last 30 days, but only include addresses that have also initiated at least one contract internal transaction during that same period. Show the address and the total value received."*

```sql
WITH recent_receivers AS (
    -- Step 1: Aggregate value received in the last 30 days
    SELECT
        to_address,
        SUM(value) AS total_value_received
    FROM `bigquery-public-data.crypto_ethereum.transactions`
    WHERE block_timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 30 DAY)
    GROUP BY to_address
),
contract_interactors AS (
    -- Step 2: Identify addresses that initiated an internal contract call
    SELECT DISTINCT from_address
    FROM `bigquery-public-data.crypto_ethereum.traces`
    WHERE block_timestamp >= TIMESTAMP_SUB(CURRENT_TIMESTAMP(), INTERVAL 30 DAY)
      AND call_type = 'call' -- specific filter found in dialect docs
)
-- Step 3: Combine and rank
SELECT
    r.to_address,
    r.total_value_received
FROM recent_receivers r
JOIN contract_interactors c ON r.to_address = c.from_address
ORDER BY r.total_value_received DESC
LIMIT 5;
```

Natural Language Query for SQL

Spider 2.0

Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows

ICLR 2025 Oral🔥

About Spider 2.0

Spider 2.0 is an evaluation framework comprising 632 real-world text-to-SQL workflow problems derived from enterprise-level database use cases. The databases in Spider 2.0 are sourced from real data applications, often containing over 1,000 columns and stored in local or cloud database systems such as BigQuery and Snowflake. This challenge calls for models to interact with complex SQL workflow environments, process extremely long contexts, perform intricate reasoning, and generate multiple SQL queries with diverse operations, often exceeding 100 lines, which goes far beyond traditional text-to-SQL challenges.

Paper | Code | Twitter | Submit

News

2024-10-29: Major update!

1. We fixed the evaluation-suite issue, so scores are now more accurate and stable, and have refreshed the

Leaderboard

Spider 2.0-Snow | Spider 2.0-DBT | Spider 2.0-lite

Spider 2.0-Snow is a self-contained text-to-SQL task that includes well-prepared database metadata and documentation, includes **547** examples, all hosted on *Snowflake*, which offers participants free quotas.
Methods with -* use special settings (ground-truth tables) and are not included in the ranking. Since we continually check the accuracy of the evaluation metrics (while the questions remain fixed), the scores may change slightly over time.

Rank	Method	Score
1 Feb 9, 2026	QUVI-3 + Gemini-3-pro-preview *DAQUV*	94.15
2 Feb 3, 2026	TCDataAgent-SQL with Contextual Scaling Engine *Tencent Cloud Big Data*	93.97
3 Jan 23, 2026	Native mini *usenative.ai*	92.50
4 Jan 27, 2026	Prism Swarm with Deepthink + Claude-Sonnet-4.5 *Paytm*	90.49

Comparative Dataset Metrics

Dataset	Complexity	Scale	SQL Dialects	Key Focus
Spider 1.0	Medium	Small	SQLite	Logic & Joins
BIRD	High	Medium	SQLite	External Knowledge
Spider 2.0	Extreme	Massive	Multi-Cloud	Workflow Agents

Specialized Benchmarks

BIRD-CRITIC (SWE-SQL): Evaluates if a model can fix buggy SQL queries

User Intent: *"Identify accounts with significant transaction variability. I need to find accounts that have made at least two transactions where the difference between their highest and lowest transaction amounts exceeds 12,000."*

The Buggy SQL (The "Issue SQL"):

```sql
SELECT
    account_id,
    MAX(payments) AS max_payment,
    MIN(payments) AS min_payment
FROM loan
GROUP BY account_id
HAVING COUNT(account_id) > 1
   AND (MAX(payments) - MIN(payments)) > 2;  -- Logical Error:
```

Filter is '2' instead of '12000'

The Task for the Agent:

1. **Identify the discrepancy:** The user asked for a difference exceeding **12,000**, but the buggy SQL uses > 2.
2. **Verify Schema/Types:** Ensure that payments is a numeric type that supports subtraction.
3. **Produce the Corrected SQL:**

Specialized Benchmarks

LogicCat: Focuses on complex reasoning (Arithmetic, Hypothetical queries).

Example Scenario: A database tracks fuel stock levels for various gas station companies.

Question: *"Assuming the fuel inventory of all Chevron gas stations suddenly decreases to 0.0001 liters, calculate the total reduction in carbon emissions."*

Answer: SELECT SUM(g.Stock_Liters - 0.0001) * 2.31 AS Carbon_Emission_Reduction_kg

```sql
SELECT
    SUM(g.Stock_Liters - 0.0001) * 2.31 AS Carbon_Emission_Reduction_kg
FROM gas g
JOIN gas_station gs ON g.Station_ID = gs.Station_ID
JOIN station_company sc ON gs.Station_ID = sc.Station_ID
JOIN company c ON sc.Company_ID = c.Company_ID
WHERE c.Company = 'Chevron';
```

Specialized Benchmarks

BiomedSQL: Domain-specific (Biomedical) tabular reasoning tasks.

Natural Language Question: "List the names of all approved drugs that target genes significantly associated with Alzheimer's Disease based on SMR analysis."

Challenges:

- **Thresholding:** "Significantly associated" in SMR typically implies a p-value threshold (e.g., $P < 0.05$).
- **Domain Filtering:** "Approved drugs" requires filtering a max_clinical_trial_phase of **4**.
- **Entity Resolution:** It must correctly link gene identifiers (like ENSG IDs) between the GWAS/SMR tables and the drug target tables.

```
SELECT DISTINCT
    d.drugName
FROM `biomedsql.knowledge_base.drug_gene_targets` d
JOIN `biomedsql.knowledge_base.smr_associations` s ON d.gene_ensg_id = s.gene_id
WHERE s.phenotype = "Alzheimer's Disease"
  AND s.p_value < 0.05                    -- Implicit: Statistical significance
  AND d.maxClinicalTrialPhase = 4         -- Implicit: "Approved" status
  AND d.drugIsApproved = TRUE;
```

Evaluation Metrics

Exact Set Match (EM): Does the SQL string match the ground truth? (Too rigid).

Execution Accuracy (EX): Does the result set of the generated SQL match the ground truth?

Relative Valid Efficiency Score (R-VES): Measures the execution speed of the query compared to the ground truth. Crucial for enterprise scalability.

Future Directions & Open Problems

- Interactive Multi-turn SQL: Refining a query through conversation ("Actually, filter that by 2024").
- Multi-modal T2Q: Querying data based on images/charts + text.
- Privacy-Preserving SQL: Generating SQL without the LLM ever "seeing" the raw data.

Text-to-NoSQL

- Traditional Text-to-SQL benchmarks (Spider, WikiSQL) assume rigid schemas.
- **Text-to-NoSQL (MongoDB, Cassandra, Neo4j, etc)** requires handling ***nested structures, graph traversal, and schemaless flexibility***

- **Key Challenges:**
  - Syntactic variety (JSON-like queries vs. Cypher).
  - Lack of large-scale annotated NoSQL datasets compared to SQL.

Specialized Domain: Document DB - Example

System Prompt:

You are an expert NoSQL developer. Convert the following natural language question into a valid MongoDB aggregation pipeline or query. Use the provided schema metadata.

Schema Metadata:

```
Collection: "orders"
Fields: {
  "_id": "ObjectId",
  "customer_id": "String",
  "items": [{"product_name": "String", "price": "Number",
"quantity": "Int"}],
  "status": "String (shipped, pending, cancelled)",
  "order_date": "ISODate"
}
```

Specialized Domain: Document DB (cont.)

Example 1 (Few-Shot):

Question: "Show me the total revenue from shipped orders."

Query:

```
db.orders.aggregate([
  { $match: { status: "shipped" } },
  { $unwind: "$items" },
  { $group: { _id: null, totalRevenue: { $sum: { $multiply:
["$items.price", "$items.quantity"] } } } }...
```

Specialized Domain: Document DB (cont.)

Example 2 (Few-Shot):

Question: "List the top 3 customers who ordered the most items in 2024."

Query:

```
db.orders.aggregate([
  { $match: { order_date: { $gte: ISODate("2024-01-01"),
$lt: ISODate("2025-01-01") } } },
  { $unwind: "$items" },
  { $group: { _id: "$customer_id", totalItems: { $sum:
"$items.quantity" } } },
  { $sort: { totalItems: -1 } },
  { $limit: 3 }      ])
```

Specialized Domain: Document DB (cont.)

Example 2 (Few-Shot):

Question: "List the top 3 customers who ordered the most items in 2024."

Thought: Filter by year 2024, group by customer_id, sum the total quantity across all items, and sort descending.

Query:

```
db.orders.aggregate([
  { $match: { order_date: { $gte: ISODate("2024-01-01"),
$lt: ISODate("2025-01-01") } } },
  { $unwind: "$items" },
  { $group: { _id: "$customer_id", totalItems: { $sum:
"$items.quantity" } } },
  { $sort: { totalItems: -1 } },
  { $limit: 3 }     ])
```

Specialized Domain: Document DB (cont.)

Current Task:

Question: "Find the names of products in orders that were cancelled, but only for customers who have more than 5 total orders."

Natural Language Query for NoSQL: DocSpider

- Without fine-tuning, performance drops considerably
- Model fine-tuning and Evaluation: A benchmark dataset is needed

Find listing ids with names in Hong Kong with no reviews from the customers

Base LLM

```
db.listings_and_reviews.find(
{"neighborhood_overview": "Hong Kong",
"number_of_reviews": {$eq:0}},
{"listing_id": 1, "name":1})
```

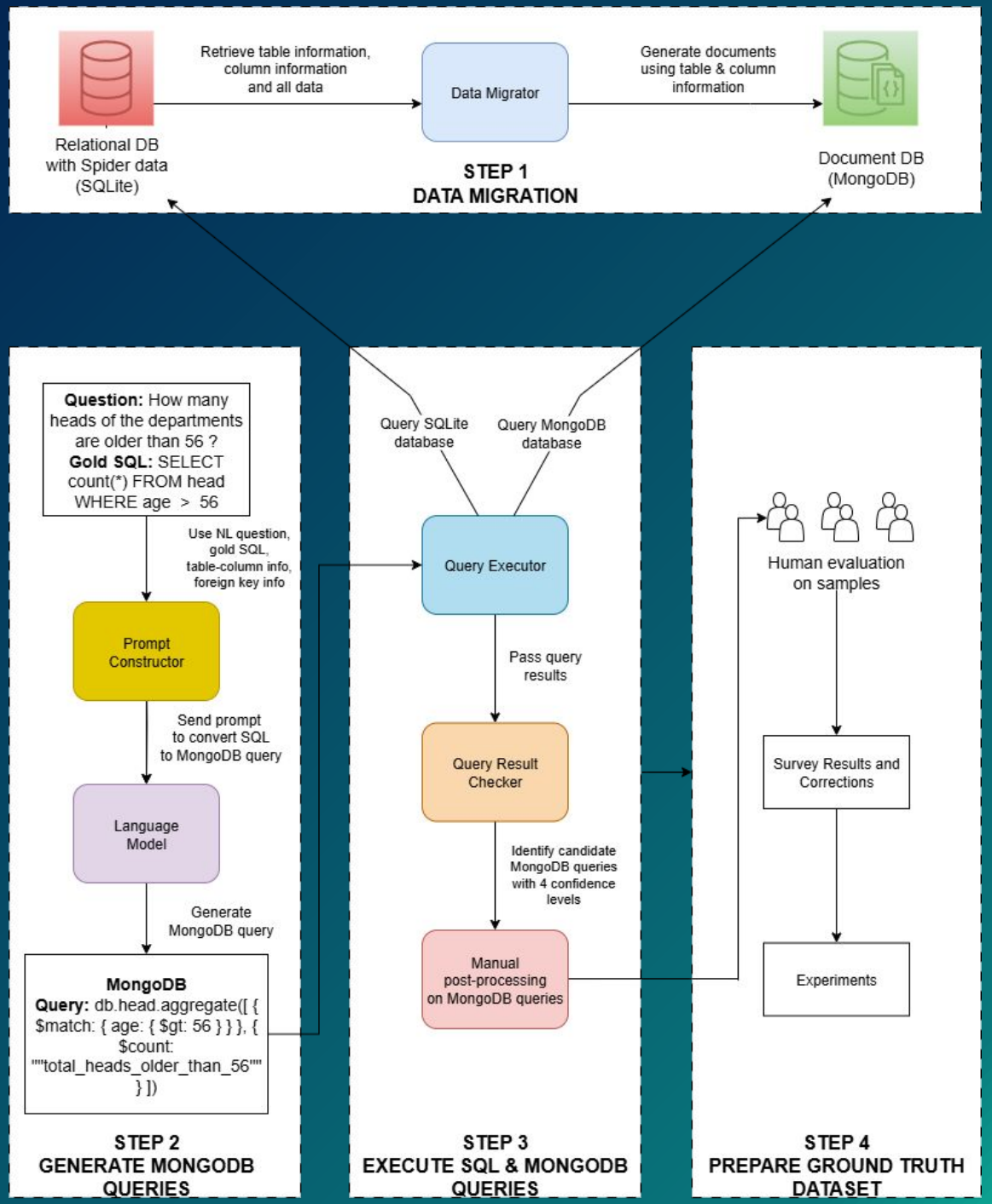
Fine-tuned LLM

```
db.listings_and_reviews.aggregate(
[ { $match: { "address.market": "Hong
Kong", "reviews": { $exists: false } }, }, {
$project: { _id: 0, listing_id: 1 } } ])
```

Paper: A.G.Ozer, R.F. Cekinel, I.H.Toroslu, P.Karagoz, DocSpider: Cross-Domain Natural Language Querying Dataset for MongoDB. *Natural Language Processing*, 1-32 (2025)
https://github.com/arifgorkemozer/docspider

NLQ for NoSQL: DocSpider

- How to provide similar capability for NoSQL databases?
- Model fine-tuning and Evaluation: A benchmark dataset is needed



NLQ for NoSQL: DocSpider

- How to provide similar capability for NoSQL databases?
- Model fine-tuning and Evaluation: A benchmark dataset is needed

Dataset	Model	Number of queries
Train	GPT4	3,670
Train	DeepSeek Coder 33B	2,818
Train-total	-	4,043
Dev	GPT4	554
Dev	DeepSeek Coder 33B	434
Dev	GPT3.5	371
Dev-total	-	620

NLQ for NoSQL: DocSpider

Table 2. : Zero-shot execution accuracy percentages

Model	Overall Exec. Accuracy	Easy	Medium	Hard	Extra Hard
Mistral 7B	20.4%	26.5%	26.4%	9.8%	0%
DeepSeek Coder 33B	29.2%	27.3%	35.8%	20.6%	17.7%
CodeLlama 70B	42.0%	43.9%	46.3%	38.2%	**26.6%**
GPT3.5	**45.4%**	**45.5%**	**53.1%**	**39.2%**	22.8%

Table 4. : Fine-tuning execution accuracy percentages

Model	Overall Exec. Accuracy	Easy	Medium	Hard	Extra Hard
Mistral 7B	40.7%	49.2%	41.4%	28.4%	39.2%
DeepSeek Coder 33B	70.8%	**84.1%**	**73.9%**	65.7%	41.8%
GPT3.5	**71.1%**	78.8%	70.4%	**70.6%**	**60.8%**

Specialized Domain: Text-to-Cypher (GraphDB)

- **Complexity:** Graph queries (Neo4j) require pathfinding logic that standard SQL does not.
- **SoTA Method: Schema-Aware Reasoning.** Providing the LLM with a schema "metadata" graph rather than just a text list of labels.
- **Current Research:** Integrating Graph Neural Networks (GNNs) with LLMs to identify the most relevant sub-graphs before query generation.

Specialized Domain: Text-to-Cypher (GraphDB)

System Prompt:

You are an expert Graph Database developer. Convert the natural language question into a valid Cypher query using the provided graph schema.

Graph Schema Metadata:

```
Nodes:
  (:User {id: String, name: String})
  (:Product {id: String, name: String, category: String})
Relationships:
  (:User)-[:PURCHASED {date: Date, rating: Int}]->
(:Product)
  (:User)-[:FOLLOWS]->(:User)
  (:Product)-[:PART_OF_BUNDLE]->(:Product)
```

Specialized Domain: Text-to-Cypher (GraphDB)

Example 1 (Few-Shot - Pathfinding):

Question: "Find all products purchased by people who 'Alice' follows."

Thought: Start with the node 'Alice', traverse the FOLLOWS relationship to other users, then traverse the PURCHASED relationship to products.

Query:

```
MATCH (u:User {name: "Alice"})-[:FOLLOWS]->(friend:User)-
[:PURCHASED]->(p:Product)
RETURN DISTINCT p.name
```

Specialized Domain: Text-to-Cypher (GraphDB)

Example 2 (Few-Shot - Recommendation Logic):

Question: "Which products are frequently bought together with the 'Turbo Blender'?"

Thought: This requires finding users who bought the 'Turbo Blender' and seeing what other products they also bought.

Query:

```
MATCH (p1:Product {name: "Turbo Blender"})<-
[:PURCHASED]-(u:User)-[:PURCHASED]->(p2:Product)
WHERE p1 <> p2
RETURN p2.name, count(*) AS frequency
ORDER BY frequency DESC
```

Specialized Domain: Text-to-Cypher (GraphDB)

Current Task:

Question: "Identify 'influencer' users who have more than 100 followers and find the most common category of products their followers have purchased."

Specialized Domain: Text-to-Cypher (GraphDB)

Key Technical Nuances for SoTA Graph Mining:

- **Variable Length Paths:** Unlike SQL joins, Cypher excels at (u:User)-[:FOLLOWS*1..3]->(other:User). LLMs must be taught when to use Kleene stars or bounded recursion.
- **Directionality:** In Cypher, (a)-[:REL]->(b) is different from (a)<-[:REL]-(b). The prompt must emphasize that the schema defines the source and target of edges.
- **Self-Loops:** LLMs often struggle with excluding the starting node in circular queries (e.g., WHERE p1 <> p2), so few-shot examples should explicitly show this safety check.

Future Trends

- **Cross-Database Reasoning:** Unified models that can generate SQL, MQL, Cypher, ElasticSearch based on the target backend
- **Streaming Text-to-NoSQL:** Real-time query generation for time-series data (e.g., InfluxDB).
- **On-Device Models:** Distilling large LLMs into smaller, privacy-preserving models for local NoSQL databases.

Selected References

- **Luo, Y., et al. (2025).** Natural Language to SQL: State of the Art and Open Problems. PVLDB, Vol. 18.
- **Li, J., et al. (2024).** BIRD: A Big Bench for Large-scale Database Grounded Text-to-SQL Evaluation. NeurIPS.
- **Zhang, X., et al. (2025).** Spider 2.0: Evaluating Language Models on Real-World Enterprise Text-to-SQL Workflows. arXiv:2411.07763.
- **Liu, Z., et al. (2025).** XiYan-SQL: A Multi-Generator Ensemble Framework for Text-to-SQL. XGenerationLab.
- **Wang, Y., et al. (2026).** AgentSM: Semantic Memory for Agentic Text-to-SQL. arXiv:2601.15709.

Selected References

- **Scholak et al. (2021):** *PICARD: Parsing Incrementally for Constrained Autoregressive Decoding from Language Models.* (Crucial for syntactic validity).
- **Li et al. (2023):** *ResSQL: Can Large Language Models Truly Do Text-to-SQL?* (Principles applicable to NoSQL).
- **Wang et al. (2020):** *RAT-SQL: Relation-Aware Schema Encoding and Corpus-Driven Multi-Task Learning.* (Foundational for graph representation).
- **Tang et al. (2024):** *A Survey on Text-to-SQL and Text-to-NoSQL in the Era of LLMs.*

Natural Language Interfaces for Database Query

Thanks for listening !

karagoz@ceng.metu.edu.tr

ODTÜ METU

EURO² EURO 4SEE