Sophrosyne: Agentic Exploration of Relational Data Systems Needs Moderation

Madhav Jivrajani
University of Illinois Urbana-Champaign

Ramnatthan Alagappan
University of Illinois Urbana-Champaign

Aishwarya Ganesan
University of Illinois Urbana-Champaign

Abstract

Text2SQL agents powered by LLMs translate natural language intent into SQL by *exploring* the data system through tool calls before formulating the query. However, to ensure secure and scoped access, data systems construct environments with explicit API surfaces. We study and categorize these APIs exposed today as either *coarse-grained* or *fine-grained* and posit that choosing between them presents a fundamental tradeoff between cost-efficient exploration and accurate SQL generation. Most data systems expose fine-grained APIs, but this inadvertently disadvantages agents: they *over-explore*, incorporating irrelevant schema elements into their query formulation and produce *inaccurate* results. We argue that curbing over-exploration is key to the effective use of these API surfaces, and propose *Sophrosyne*, a data system environment that augments API responses with *directives* that guide the agent's exploration process. Initial results show that directives reduce over-exploration by 4.6× and boost accuracy by up to 12.4% (~4 percentage points).

CCS Concepts

• **Computing methodologies** → **Artificial intelligence**;
• **Information systems** → **Middleware for databases**.

Keywords

data agents, text2SQL, model context protocol

1 Introduction

With advanced tool calling abilities [39], AI agents are emerging as the primary users of data systems [24, 46], and text2SQL agents in particular offer a promising approach to translating natural language queries to SQL. Agents begin by *exploring* [24] the data system to gather relevant tables, columns, and `JOIN` relationships prior to formulating the final query.

While exposing data systems to agents is powerful, securing and scoping access to them is crucial [14, 28]. As a result, data systems construct environments with explicit API surfaces and authentication [8–11, 19, 20, 27, 29, 30, 35, 42, 45] using protocols such as MCP [4]. Agents execute tool calls by invoking APIs exposed to it as shown in Figure 1. Since this is becoming the ubiquitous way data systems are exposed to agents, we ask the question: *how does the exposed API surface effect the agent's ability to perform text2SQL tasks?* We conduct the first study of API surfaces exposed by different data systems and find that they fall into two broad categories and differ primarily in how they enable exploration: *coarse-grained* and *fine-grained* APIs. *Coarse-grained* APIs allow agents to retrieve the entire schema up-front for the LLM to process in its entirety and determine relevant parts for query formulation. *Fine-grained* APIs enable progressive exploration, starting with just table names and allow the agent to selectively inspect individual table schemas.

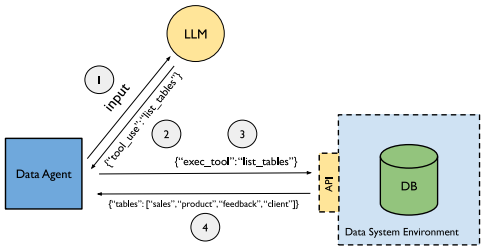


Figure 1: Agents and Data System Environments

Based on our study, we observe a dichotomy between these choices. With coarse-grained APIs, the agent gets all the schema information, and has to then *distill* down what is relevant. However, this leads to increased token cost (§2.3) during exploration. Fine-grained APIs enable a more token-efficient exploration process, but the agent has no visibility into the full schema. Since the relevant tables are not known *a priori*, constructing a correct query *necessitates* exploring broadly, causing the agent to *over-explore* and incorporate irrelevant tables into query construction, ultimately degrading accuracy. Furthermore, we summarize and show in Table 1 that a majority of data systems expose fine-grained APIs, making any agent that is exposed to them prone to over-exploration and inaccurate SQL generation.

Our key insight for the effective use of fine-grained APIs is simple: let the environment provide the agent with *directives* that help constrain its future exploration. Since over-exploration stems from exploring tables that later prove irrelevant, directives steer the agent away from such tables. Our initial results show that directives reduce over-exploration by up to 4.6× and boost SQL generation accuracy by up to 12.4% (~4 percentage points). A key challenge is computing these directives and we present *Sophrosyne*, a data system environment that addresses it.

2 Motivation

We now analyze the inefficiencies of coarse-grained API surfaces and quantify the problem of over-exploration.

Sophrosyne is the ancient Greek virtue of prudence and moderation

Table 1: Categorization of API Surfaces

API Surface	Data System
Coarse-grained	Google Spanner [20], PlanetScale [35]
Fine-grained	Amazon DSQL [8], Amazon MySQL [9], Amazon Postgres [10], Amazon Redshift [11], GCP BigQuery [19], Azure SQL [27], Supabase [42], Neon [30], MotherDuck DuckDB [29], Turso [45]

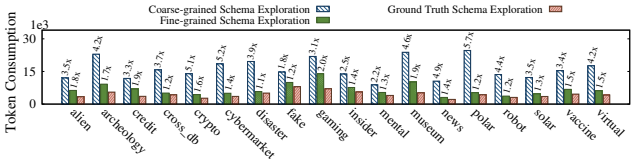


Figure 2: Inefficiencies of coarse-grained exploration

2.1 Setup

Our data system environment is an MCP server for SQLite [21] exposing the fine-grained API surface described in Table 2. We build our agent (§A.2) using OpenCode [3], and evaluate it on 157 queries from the BIRD LiveSQLBench benchmark [44]. We evaluate with three models: GPT-5.4-mini, GPT-5.4, and Claude Sonnet 4.5, all configured using their default reasoning effort unless otherwise specified.

2.2 Extent of Over-exploration

To measure over-exploration, we analyze the agent's tool call sequences. A tool call is *exploratory* if it inspects a table's schema via `describe_table` or a `read_query` call of the form `PRAGMA table_info (table_name)`. We establish the ground truth exploration count by parsing each ground truth SQL query and extracting the number of unique tables it references, representing the exact number of schema explorations needed. Furthermore, to understand the effect a model's reasoning effort has on over-exploration, we measure it at each model's highest reasoning effort as well.

Figure 3 shows the percentage of exploratory calls across all queries above the ground truth. When exposed to fine-grained API surfaces, agents over-explore by up to 65%, with reasoning effort having a negligible impact on over-exploration, suggesting that for each model, it is an inherent symptom when exposed to fine-grained API surfaces. More critically, over-exploration translates to inaccuracies in query construction as we discuss in §4.1.

2.3 Inefficiencies of Coarse-grained API Surfaces

Since token consumption directly proxies cost, we measure the inefficiency of coarse-grained surfaces using Sonnet-4.5. With coarse-grained exploration, since the agent retrieves the entire schema for every query, we estimate its token consumption as the entire schema's token count times the number of queries. For comparison, we compute (1) the actual schema token consumption as the sum of schema tokens of tables used in the ground truth queries provided by BIRD, and (2) the total schema tokens of tables explored when the agent is exposed to a fine-grained API surface. Figure 2 compares these per database provided. Coarse-grained exploration consumes up to 5.7× more tokens than necessary while fine-grained exploration consumes up to 2× more, making coarse-grained exploration APIs highly cost-inefficient.

Table 2: API surface exposed by *Sophrosyne*

API	Description
`list_tables`	Return names of tables in the database
`describe_table`	Return schema of a table
`get_join_info`	Return foreign key relationships between tables or for a specific table
`read_query`	Execute read-only queries for data exploration and error validation
`submit_query`	Submit the final SQL query

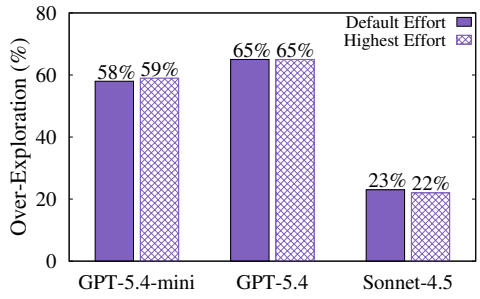


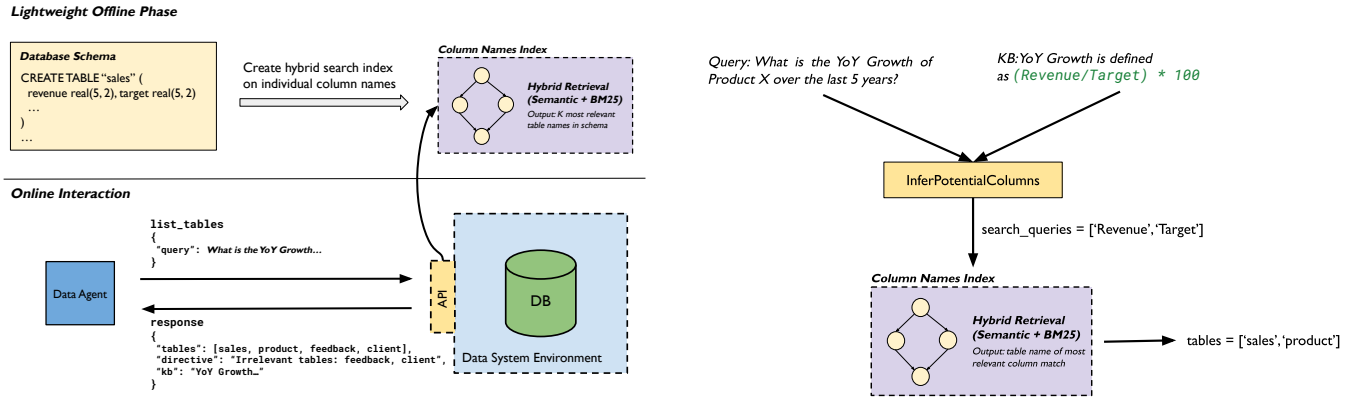
Figure 3: Over-exploration in data agents

3 Design & Implementation

Over-exploration stems from exploring tables that prove to be irrelevant. Our key idea to address this issue is to compute and return *directives* that convey the set of potentially irrelevant tables as feedback to the agent to help guide its future exploration (as shown in Figure 4a). The central challenge here is computing these directives. We design and implement *Sophrosyne*, a data system environment that solves this.

Directives are returned with the response of `list_tables`, since we observe this to consistently be the first tool call made by the agent. Interaction with *Sophrosyne* requires a lightweight offline phase that constructs a hybrid search index ($\mathbf{HI_{col}}$) over individual column names, combining an IVF index for semantic similarity with a BM25 index [38] for keyword matches. The results are combined using the reciprocal rank fusion method [15]. To generate embeddings for $\mathbf{HI_{col}}$, we use OpenAI's `text-embedding-3-large` model. As a result, $\mathbf{HI_{col}}$ can be used to search for relevant column names and which table contains them. On searching $\mathbf{HI_{col}}$, it returns the table name of the most relevant match.

The key idea behind directives is that for the knowledge



(a) *Sophrosyne* and the *directive* mechanism **(b) Computing directives from knowledge base entries**

Figure 4: Overview of *Sophrosyne*'s design

sources provided by the BIRD benchmark, directives are computed based on the observation that entries are defined in terms of relevant column names. Querying $\mathbf{HI_{col}}$ using the entire knowledge base entry can lead to inaccuracies given multiple column names are likely to be present [26]. As a result, we first infer *potential* column names by prompting (§A.1) an inexpensive model as shown in Figure 4b. The inferred names now serve as search queries for $\mathbf{HI_{col}}$. Running retrieval for each of these gives us a set of potentially *relevant* tables for this query. Directives are now the *irrelevant* tables, computed as all table names that are *not* part of this set.

Sophrosyne is implemented as an MCP server following current data systems [8–11, 19, 27, 29, 30, 42] and exposes the fine-grained API surface laid out in Table 2. We posit that directive computation should be tailored to the data system and the available context and that better mechanisms will further enhance the agent's success. *Sophrosyne* balances offline and online costs and while schema linking approached [12, 43] make different cost tradeoffs, they can help achieve the same goal. Furthermore, companies building data agents [2, 16, 33] can leverage the semantically richer context available to them to compute directives more effectively.

4 Evaluation

We use the setup described in §2.1, with our agent overriding OpenCode's system prompts (§A.2). We also disallow use of built-in tools thereby treating OpenCode as a simple agent loop with access to only the API surface exposed to it. Consequently, any agent implementation with MCP support [3, 5, 7, 32, 41] can be used. Additionally, BIRD provides domain-specific knowledge bases to provide the agent with prerequisite context. Since retrieval of knowledge entries can be inaccurate [26], we provide the agent with the exact entries needed for each query to isolate its effects and compare the following three systems on the basis of the directive mechanism: *No Directives*, *Sophrosyne*, *Sophrosyne-Oracle*.

No Directives: This is a faithful representation of existing data system environments that expose fine-grained API surfaces. It exposes the API surface laid out in Table 2.

Sophrosyne: This also exposes the API surface laid out in Table 2, computes directives and returns them as part of the response for `list_tables` as described in §3.

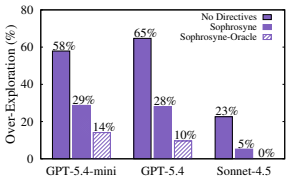
Sophrosyne-Oracle: This is identical to *Sophrosyne* except it assumes an oracle present for computing fully accurate directives. The system looks at ground truth SQL queries to compute the set of irrelevant tables thereby guaranteeing 100% accurate directives. With these, *Sophrosyne-Oracle* intends to represent the upper bound on agent performance when exposed to a fine-grained API surface.

4.1 Over-exploration & Its Effects

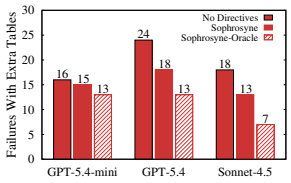
We measure exploration using the methodology in §2.2. Figure 5a shows over-exploration as a percentage of exploratory calls above the required number. With directives, *Sophrosyne* consistently reduces over-exploration across all models, and *Sophrosyne-Oracle* reduces it further, indicating that more accurate directive computation only yields better results.

Next, we examine how directives and over-exploration affect query accuracy. Figure 5b shows the number of failed queries that reference tables absent from the ground truth. *No Directives* serves as the baseline for these failures, though not all can be attributed to over-exploration, as logical errors in query formulation are still possible. With *Sophrosyne*, these failures decrease consistently across all models. While we get improved accuracy with imperfect directives (§4.3), *Sophrosyne-Oracle* further improves it with perfect directives indicating the effect over-exploration has on query accuracy.

For Sonnet-4.5, while *Sophrosyne-Oracle* eliminates over-exploration as seen in Figure 5a, we still see failures with extra tables (Figure 5b) because although the agent may not explore these tables using explicit tool calls, it is still made aware of them via information returned by `get_join_info`.



(a) Measure of over-exploration



(b) Failures with extra tables

Figure 5: Quantifying over-exploration and its effects

Table 3: Execution Accuracy and Token Cost

Model	System	Exec. Accuracy	Input Tokens	Output Tokens
GPT-5.4-mini	*No Directives*	37.6%	7.67M	**585K**
	Sophrosyne	**39.5%**	**7.60M**	592K
	Sophrosyne-Oracle	**41.4%**	**6.67M**	**557K**
GPT-5.4	*No Directives*	30.6%	6.28M	406K
	Sophrosyne	**34.4%**	**6.15M**	**404K**
	Sophrosyne-Oracle	**37.6%**	**5.04M**	**383K**
Sonnet 4.5	*No Directives*	41.4%	8.74M	343K
	Sophrosyne	**42.7%**	**8.00M**	**318K**
	Sophrosyne-Oracle	**45.9%**	**6.72M**	**310K**

4.2 Execution Accuracy & Token Cost

We use token counts as a proxy for inference cost [6, 34]. Table 3 summarizes execution accuracy, input tokens and output tokens (including reasoning) across the three systems. *Sophrosyne* consistently improves accuracy by reducing over-exploration, and since fewer schema elements are explored, the corresponding token counts also reduce. GPT-5.4-mini is an exception, incurring a 1.2% output token overhead due to excess reasoning tokens and the model summarizing verbose SQL results before terminating for certain queries. We discuss cost in more detail in §4.3. *Sophrosyne-Oracle* consistently improves execution accuracy and improves token cost, further indicating that better mechanisms to compute directives will only lead to better agent performance.

4.3 Directive Accuracy & System Overhead

We now discuss the accuracy and overhead of our directive mechanism (§3). The offline phase embeds column names, costing *one-tenth of a cent* totally. We treat this as negligible and ignore it in our discussion ahead. To measure accuracy of computed directives we calculate recall percentage with respect to the ground truth set of tables, and across runs of the agent, this averages to **82.5%**.

Table 4 summarizes the base cost and overhead incurred with *Sophrosyne*. The overhead incurred is *8 cents*, covering cost to infer columns using GPT-5.4-nano and subsequently embed them (§3). *Sophrosyne* is more cost-efficient except for GPT-5.4-mini, where the base cost reduction does not absorb the overhead. Sonnet-4.5 sees the largest gains, consistent with the over-exploration reduction in Figure 5a.

Table 4: Overhead of directive computation in *Sophrosyne*

Model	System	Base Cost	Over-head	Total
GPT-5.4-mini	*No Directives*	$8.39	-	**$8.39**
	Sophrosyne	$8.36	$0.08	$8.44
GPT-5.4	*No Directives*	$21.79	-	$21.79
	Sophrosyne	$21.43	$0.08	**$21.51**
Sonnet-4.5	*No Directives*	$31.36	-	$31.36
	Sophrosyne	$28.80	$0.08	**$28.88**

5 Related Work

Text2SQL has a rich body of work ranging from building and fine-tuning custom models [18, 23, 31, 36, 37], to using LLMs with sophisticated prompting strategies [40, 43] and inference time scaling [17] to improve the accuracy of text2SQL. With LLMs becoming more adept at tool calling and interacting with data systems, works on agentic text2SQL focus on improving accuracy by synthesizing common failure modes in an offline phase and modifying the runtime workflow to provide relevant context to the agent [1, 13]. Our work is the first to study the effect of the exposed API surface on text2SQL accuracy and identify over-exploration as a symptom of an overlooked tradeoff in API design. *Sophrosyne* addresses this by augmenting API responses, and can therefore be used with any existing agent and workflow, as well as in conjunction with other works, further improving accuracy.

6 Future Work

We now outline directions for future work. Even with perfect directives, some models still over-explore, while those that avoid over-exploration can still produce failed queries with extra tables (Figure 5) suggesting the effectiveness of directives depends on the model's instruction-following ability. Since directives are returned early, their influence may decay as the agent's context grows [22] and returning *reminders* when directives are not followed could help keep the model in line. Next, data agents answering queries spanning multiple data sources [25] need to discover these sources and subsequently relevant schema elements across them. With fine-grained surfaces, the effects of over-exploration would only be exacerbated. Directives aware of multiple sources would prove essential in this setting. Finally, in addition to guiding the exploration process, directives can also provide *performance hints* to the agent in order to construct more efficient SQL. For example, nudging the agent to use a composite index's column prefix can dramatically improve query performance for large tables.

References

[1] Shubham Agarwal, Asim Biswal, Sepanta Zeighami, Alvin Cheung, Joseph Gonzalez, and Aditya G. Parameswaran. Arming data agents with tribal knowledge. https://arxiv.org/abs/2602.13521, 2026.

[2] Analytics at Meta. Inside meta's home grown ai analytics agent. https://medium.com/@AnalyticsAtMeta/inside-metas-home-grown-ai-analytics-agent-4ea6779acfb3, March 2026.

[3] Anomaly (SST). OpenCode: The open source AI coding agent. https://opencode.ai, 2025. Apache-2.0 license. Accessed: 2026-03-15.

[4] Anthropic. Introducing the Model Context Protocol. https://www.anthropic.com/news/model-context-protocol, November 2024. Accessed: 2025-12-11.

[5] Anthropic. Claude code. https://code.claude.com/docs/en/overview, 2025. Accessed: 2026-03-15.

[6] Anthropic. Pricing - Claude documentation. https://docs.claude.com/en/docs/about-claude/pricing, 2025. Accessed: 2025-12-11.

[7] Anysphere. Cursor: The AI code editor. https://cursor.com, 2024. Accessed: 2026-03-15.

[8] AWS Labs. Aurora DSQL MCP server. https://awslabs.github.io/mcp/servers/aurora-dsql-mcp-server, 2025. Apache-2.0 license. Accessed: 2026-04-19.

[9] AWS Labs. MySQL MCP server. https://awslabs.github.io/mcp/servers/mysql-mcp-server, 2025. Apache-2.0 license. Accessed: 2026-04-19.

[10] AWS Labs. Postgres MCP server. https://awslabs.github.io/mcp/servers/postgres-mcp-server, 2025. Apache-2.0 license. Accessed: 2026-04-19.

[11] AWS Labs. Redshift MCP server. https://awslabs.github.io/mcp/servers/redshift-mcp-server, 2025. Apache-2.0 license. Accessed: 2026-04-19.

[12] Muhammad Imam Luthfi Balaka, David Alexander, Qiming Wang, Yue Gong, Adila Krisnadhi, and Raul Castro Fernandez. Pneuma: Leveraging LLMs for tabular data representation and retrieval in an end-to-end system. *Proc. ACM Manag. Data*, 3(3), June 2025.

[13] Asim Biswal, Chuan Lei, Xiao Qin, Aodong Li, Balakrishnan Narayanaswamy, and Tim Kraska. AgentSM: Semantic memory for agentic text-to-SQL. https://arxiv.org/abs/2601.15709, 2026.

[14] Claude Platform. Building agents that reach production systems with mcp. https://claude.com/blog/building-agents-that-reach-production-systems-with-mcp, April 2026.

[15] Gordon V. Cormack, Charles L A Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '09, page 758–759, New York, NY, USA, 2009. Association for Computing Machinery.

[16] Databricks. Introducing genie agent mode. https://www.databricks.com/blog/introducing-genie-agent-mode, April 2026.

[17] Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. ReFoRCE: A text-to-SQL agent with self-refinement, consensus enforcement, and column exploration. https://arxiv.org/abs/2502.00675, 2025.

[18] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, January 2024.

[19] Google Cloud. Google cloud MCP servers: Big query. https://docs.cloud.google.com/bigquery/docs/reference/mcp/tools_overview, 2025. Preview. Accessed: 2026-03-15.

[20] Google Cloud. Use the Spanner remote MCP server. https://docs.cloud.google.com/spanner/docs/use-spanner-mcp, 2026. Accessed: 2026-03-15.

[21] Richard D. Hipp. SQLite. https://www.sqlite.org/index.html, 2020. Version 3.31.1.

[22] Kelly Hong, Anton Troynikov, and Jeff Huber. Context rot: How increasing input tokens impacts LLM performance. Technical report, Chroma, July 2025.

[23] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3), May 2024.

[24] Shu Liu, Soujanya Ponnapalli, Shreya Shankar, Sepanta Zeighami, Alan Zhu, Shubham Agarwal, Ruiqi Chen, Samion Suwito, Shuo Yuan, Ion Stoica, Matei Zaharia, Alvin Cheung, Natacha Crooks, Joseph E. Gonzalez, and Aditya G. Parameswaran. Supporting our AI overlords: Redesigning data systems to be agent-first. https://arxiv.org/abs/2509.00997, 2025.

[25] Ruiying Ma, Shreya Shankar, Ruiqi Chen, Yiming Lin, Sepanta Zeighami, Rajoshi Ghosh, Abhinav Gupta, Anushrut Gupta, Tanmai Gopal, and Aditya G. Parameswaran. Can ai agents answer your data questions? a benchmark for data agents, 2026.

[26] Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. Query rewriting for retrieval-augmented large language models. https://arxiv.org/abs/2305.14283, 2023.

[27] Microsoft. Azure SQL MCP Tools. https://learn.microsoft.com/en-us/azure/data-api-builder/mcp/data-manipulation-language-tools, 2025. Accessed: 2026-04-19.

[28] Model Context Protocol. Building agents that reach production systems with mcp. https://modelcontextprotocol.io/docs/tutorials/security/authorization.

[29] MotherDuck. MotherDuck's DuckDB MCP server. https://github.com/motherduckdb/mcp-server-motherduck, 2025. MIT license. Accessed: 2026-04-19.

[30] Neon Database. Mcp server for interacting with neon management api and databases. https://github.com/neondatabase/mcp-server-neon, 2026. MIT license. Accessed: 2026-03-15.

[31] Xuan-Bang Nguyen, Xuan-Hieu Phan, and Massimo Piccardi. Fine-tuning text-to-sql models with reinforcement-learning training objectives. *Natural Language Processing Journal*, 10:100135, 2025.

[32] OpenAI. Codex CLI. https://developers.openai.com/codex/cli/, 2025. Accessed: 2026-03-15.

[33] OpenAI. Inside our in-house data agent. https://openai.com/index/inside-our-in-house-data-agent/, January 2026.

[34] OpenAI. OpenAI API pricing. https://developers.openai.com/api/docs/pricing, 2026. Accessed: 2026-04-02.

[35] PlanetScale. PlanetScale MCP server. https://github.com/planetscale/mcp-server, 2026. Apache-2.0 license. Accessed: 2026-03-15.

[36] Mohammadreza Pourreza and Davood Rafiei. DTS-SQL: Decomposed text-to-SQL with small large language models. In Yaser Al-Onaizan, Mohit Bansal, and Yun-Nung Chen, editors, *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA, November 2024. Association for Computational Linguistics.

[37] Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. ROUTE: Robust multitask tuning and collaboration for text-to-SQL. In *The Thirteenth International Conference on Learning*

Representations, 2025.

[38] Stephen E. Robertson and Steve Walker. Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 232–241, Dublin, Ireland, 1994. Springer-Verlag.

[39] Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. https://arxiv.org/abs/2302.04761, 2023.

[40] Vladislav Shkapenyuk, Divesh Srivastava, Theodore Johnson, and Parisa Ghane. Automatic metadata extraction for text-to-SQL. https://arxiv.org/abs/2505.19988, 2025.

[41] Sourcegraph. Amp: AI coding agent. https://ampcode.com, 2025. Accessed: 2026-03-15.

[42] Supabase Community. Supabase MCP server. https://github.com/supabase-community/supabase-mcp, 2025. Apache-2.0 license. Accessed: 2026-03-15.

[43] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: Contextual harnessing for efficient SQL synthesis. https://arxiv.org/abs/2405.16755, 2024.

[44] BIRD Team. LiveSQLBench: A dynamic and contamination-free benchmark for evaluating LLMs on real-world text-to-SQL tasks. https://github.com/bird-bench/livesqlbench, 2025. Accessed: 2025-05-22.

[45] Turso. Introducing the Turso database MCP. https://turso.tech/blog/introducing-the-turso-database-mcp-server, 2025. Accessed: 2026-05-03.

[46] Matei Zaharia. Bridging the operational and analytical worlds with Lakebase. In *Proceedings of the 51st International Conference on Very Large Data Bases*, VLDB '25, 2025. Keynote 3.

A System Prompts

A.1 Column Inference System Prompt

Below is the system prompt used for column inference in the directive computing mechanism (§3).

Column Inference System Prompt

You extract column-search terms for schema/table filtering.

Task: Given a natural-language database question and optional knowledge-base context for one fact or formula, return a JSON array of short phrases that are most likely to match column names, column descriptions, or column meanings. These phrases will be used to retrieve relevant columns and tables.

Primary objective: Maximize recall of relevant column concepts without adding generic SQL noise.

Include:

- Domain metrics, measures, statuses, categories, thresholds, units, dimensions, or attributes needed to answer the question
- Concepts used for filtering, grouping, ordering, joining, or computing the answer
- Variables or operands explicitly named in the KB context if they would need backing columns
- Specific qualified concepts rather than broad parents
- A likely schema-style variant only when it is materially different and useful

Do not include:

- SQL operation words such as `count`, `sum`, `average`, `max`, `min`, `top`, `sort`, `group`
- Generic verbs or filler words such as show, list, find, calculate, value, record, row
- Table names or entity names unless they are also plausible column concepts
- Duplicated broad + specific pairs; keep the more specific phrase
- Explanations, labels, or any text outside the JSON array

Normalization rules:

- Output 0 to 8 items
- Order from most important to least important
- Use short lowercase noun phrases
- Prefer singular forms when natural
- No punctuation except meaningful abbreviations
- Return only raw JSON

Examples

Question: Which observations have approximate air quality above safe levels?
Output: `["approximate air quality", "safe level"]`

Question: Find the largest circumference.
Context: circumference = 2 × CircleConstant × Radius / UnitOfMeasure
Output: `["circumference", "circle constant", "radius", "unit of measure"]`

Question: Show facilities with severe corrosion risk.
Output: `["severe corrosion risk"]`

Return only the JSON array.

A.2 Agent System Prompt

Below is the system prompt using which the OpenCode agent is constructed.

Agent System Prompt

You are a confident data analyst assistant with access to a SQLite database. Your task is to answer the user's question by querying the database.

Available tools:

- `list_tables`: List all tables in the database.
- `describe_table`: Get schema information for a single table.
- `get_join_info`: Get all foreign key relationships between tables to understand how tables relate to each other.
- `read_query`: Execute a SELECT query for exploration and intermediate checks.
- `submit_final_query`: Submit your final SQL query that answers the user's question. Only call this when you are confident in your answer.

Strategy:

(1) Execute appropriate queries to find the information needed.
(2) Make sure the queries you issue don't return massive intermediate outputs since that might exhaust the context window.
(3) Provide a clear, concise answer to the user's query, provide only what is asked. No explanations. No insights.

Critical:

(1) Use `max_rows` whenever possible with `read_query` to minimise the amount returned by the server.
(2) Use ONLY SQL queries to arrive at the final result. All formatting should happen via SQL. No processing.
(3) You MUST use `submit_final_query` and ONLY for your final answer query.
(4) Once you are instructed to terminate, do so immediately.

Be methodical in your exploration.