Thucy: An LLM-based Multi-Agent System for Claim Verification across Relational Databases

Michael Theologitis[1], **Dan Suciu**[1]

[1]University of Washington
Paul G. Allen School of Computer Science & Engineering
{mthe, suciu}@cs.washington.edu

Abstract

In today's age, it is becoming increasingly difficult to decipher truth from lies. Every day, politicians, media outlets, and public figures make conflicting claims—often about topics that can, in principle, be verified against structured data. For instance, statements about crime rates, economic growth or healthcare can all be verified against official public records and structured datasets. Building a system that can automatically do that would have sounded like science fiction just a few years ago. Yet, with the extraordinary progress in LLMs and agentic AI, this is now within reach. Still, there remains a striking gap between what is technically possible and what is being demonstrated by recent work. Most existing verification systems operate only on small, single-table databases—typically a few hundred rows—that conveniently fit within an LLM's context window.

In this paper we report our progress on THUCY, the first cross-database, cross-table multi-agent claim verification system that also provides concrete evidence for each verification verdict. THUCY remains completely agnostic to the underlying data sources before deployment and must therefore autonomously discover, inspect, and reason over all available relational databases to verify claims. Importantly, THUCY also reports the exact SQL queries that support its verdict (whether the claim is accurate or not) offering full transparency to expert users familiar with SQL. When evaluated on the TabFact dataset—the standard benchmark for fact verification over structured data—THUCY surpasses the previous state of the art by 5.6 percentage points in accuracy (94.3% vs. 88.7%).

Code — https://github.com/michaeltheologitis/thucy

1 Introduction

In the Annual Report released last year by the Seattle City Attorney's Office (2024), we read the following:

> *I am pleased to acknowledge that 2024 saw a reduction in property crime and violent crime in Seattle.*
> *— Ann Davison, City Attorney*

However, for many residents of Seattle, this statement might not quite match their lived experience. The natural instinct is to want to find out more. Was crime really down in 2024? And if so, by how much—and according to which source?

It turns out, the City of Seattle publicly provides an official crime dataset (Seattle Police Department 2025b)—with all crimes from 2008 until now—that is structured, detailed, and updated almost daily. In principle, that is all you would need to verify such a claim. In practice, though, very few people ever try. Most will simply take the statement at face value and move on, keeping the comforting thought that "Seattle is safer now" somewhere in the back of their mind.

A few more curious and determined souls might go a step further, dig around, discover the dataset, and even download it. Then reality hits: it is technical, messy, and not exactly friendly to non-specialists. So they, too, eventually give up. And so the claim remains—unchecked, unchallenged, and protected by the technical complexity of verification.

In this work, we present a multi-agent system called THUCY that takes over the verification process once the user has obtained the structured data and imported it into a relational database. From that point on, THUCY figures everything out: it autonomously explores the available data sources, reasoning over them on the fly to produce a verdict and supporting evidence.

In our example, we can simply download the City of Seattle's official crime dataset, load it into a SQL database, and invoke THUCY with the verbatim claim of Ann Davison for verification. THUCY takes care of the rest—no further clarifications are needed. In fact, THUCY is completely agnostic to the underlying data environment before deployment.

We draw inspiration from the work of Thucy(dides), the Athenian historian (460–400 BC) who wrote the *History of the Peloponnesian War* between Sparta and Athens. "Thucydides has been dubbed the father of *scientific history* by those who accept his claims to have applied strict standards of impartiality and evidence-gathering" (Wikipedia 2025).

Following Thucydides' example of reporting, THUCY's job is twofold: ① provide a verification verdict (whether the claim is supported or not based on the available data), and ② return a report together with SQL queries that explain its findings. By returning the explanations in the form of SQL queries, THUCY empowers the data analyst to modify these SQL queries and explore the claim further. For example they can "roll-up" by checking if crime of all types has decreased in Seattle in 2024 (not just property and violent crime), or to "drill down" and check how crime changed in 2024 for each Seattle neighborhood.

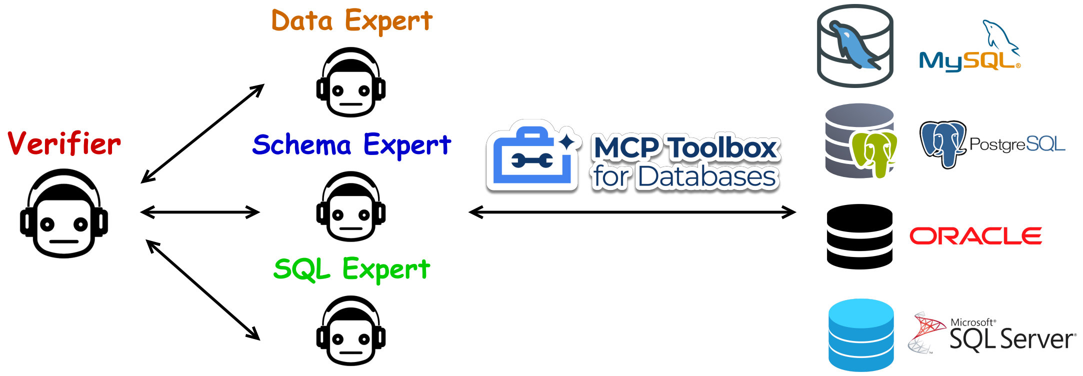

Figure 1: The architecture of THUCY, a multi-agent system led by the *Verifier*. Its job is to verify NL claims grounded in relational databases and report the corresponding SQL evidence. The *Verifier* coordinates three expert agents: the *Data Expert*, which summarizes available data sources; the *Schema Expert*, which answers schema-related questions; and the *SQL Expert*, which writes and executes SQL queries to obtain verifiable answers. The data layer follows a plug-and-play design and can include any number of relational databases—each potentially containing many tables—with PostgreSQL, MySQL, SQL Server, and Oracle shown here only as examples. THUCY remains fully agnostic to the underlying data sources. The agents must therefore operate in an open-ended environment, discovering and reasoning about available data as they encounter it. The experts interact with these relational databases through specialized tools managed via Google's MCP Toolbox. Adding or removing databases is straightforward: it simple involves adding or removing the corresponding tool from the toolbox.

2 Architecture

In this section, we describe the architecture of THUCY. We start by discussing the data sources and our minimal assumptions about them. Then, we go over the recent standardized ways modern AI agents connect to databases. Finally, we delve into the details of our multi-agent system (Figure 1).

Throughout this section, we aim to be as informative as possible about the unique ways multi-agent systems must navigate relational databases. Table 1 summarizes, at a high level, how THUCY differs from prior systems that operate over structured data. Beyond explaining how our system works, our goal is to also make clear the rationale behind the design choices that made THUCY possible. Doing so naturally requires unpacking some of the subtleties of relational databases along the way.

Data Sources

The vision behind THUCY is simple. A user can drop a few *grounding* data sources into SQL databases and immediately start asking the system to verify claims. We make only minimal assumptions about these data sources: they are *relational*—as is often the case with official federal or state data—and we treat them as reliable and trustworthy.

THUCY remains completely agnostic of both the information content and the internal structure of these tables and databases. We provide no additional metadata, schema information, or prior knowledge to our multi-agent system. Instead, we assume that the *grounding* data sources are entirely unknown before deployment. The agents must therefore operate in an open-ended environment, discovering and reasoning about available data as they encounter it.

This design makes our approach highly flexible as we can quickly plug-and-play by adding or removing data sources without concern for compatibility or reconfiguration. This flexibility has been a central motivation since the inception of our system.

Tools

To enable this flexibility, we must address a fundamental challenge: LLMs, no matter how capable, are inherently disconnected from external data sources—they can only operate in isolation, with no way to interact with databases. Agents bridge this gap by using *tools*. A tool acts as an interface to external capabilities, allowing agents to interact with, perceive, and affect their environment. In general, tools can include capabilities that perform mathematical calculations, or read files from disk, or query a database. Each agent has a fixed collection of such tools. At runtime, the agent[1] autonomously decides which tool to invoke, how to call it, and when to use it; the tool's output is then fed back into the reasoning loop (Yao et al. 2023). This interactive feedback cycle between reasoning, action, and observations forms the backbone of modern agentic AI.

Building tools from scratch is challenging, because they need to be carefully designed. They must return well formatted values, and informative error messages, because these are fed back into the LLM. Building a tool also requires domain expertise. For example, a simple tool that fetches schema information from a PostgreSQL database requires knowledge of relational databases, Postgres internals, and query execution. Building a similar tool for MySQL (another database management system), the developer has to

[1] More precisely, it is the LLM that makes this decision, though we often use "agent" and "LLM" interchangeably in such contexts

Table 1: Capabilities of different LLM-based systems for *fact verification* over structured data. Cross-Table and Cross-Database refer to a system's ability to verify claims that span multiple tables or databases. *Interpretable* means that users can understand the reasoning behind the model's verdict. *Verifiable* goes a step further—it allows users to reproduce the verification process (e.g., providing the exact Python or SQL commands), eliminating any suspicion of hallucinations. Finally, *Source-Agnostic* indicates that the system can operate without prior knowledge of its data environment, figuring out everything from scratch.

Method	**Cross-Table**	**Cross-Database**	**Interpretable**	**Verifiable**	**Source-Agnostic**
BINDER (Cheng et al. 2023)	✗	✗	✓	✗	✗
DATER (Ye et al. 2023)	✗	✗	✓	✗	✗
CoTable (Wang et al. 2024)	✗	✗	✓	✗	✗
ReAcTable (Zhang et al. 2024)	✗	✗	✗	✗	✗
AutoTQA (Zhu et al. 2024)	✓	✓	✗	✗	✗
POS (Nguyen et al. 2025)	✗	✗	✓	✗	✗
THUCY (ours)	✓	✓	✓	✓	✓

start over, since there are differences in the catalog layout, the connection logic, etc. Switching to a different agentic framework might require rebuilding all tools from scratch. The solution we adopted for THUCY was to use MCP.

MCP The Model Context Protocol (MCP), introduced by Anthropic (2024), standardizes how AI applications connect to different data sources, effectively eliminating the need for custom connections for each new AI model and external system allowing us to direct our energy elsewhere—away from repetitive boilerplate code. MCP simplifies and streamlines the tool building process, and has already been adopted by industry (Mehrotra 2025; Gonzales and Murching 2025; Ganguly 2025; Agarwal et al. 2025).

Toolsets We use Google's MCP Toolbox for Databases (Buvaraghan and Egan 2025), a framework that makes it effortless to organize and manage database tools. It provides built-in *primitives*—actual implementations of low-level functions like executing SQL—across different database systems (e.g., PostgreSQL, MySQL), ready to use without us having to code anything.

Using these primitives as building blocks, we define higher-level tools that *bind* and interact with specific databases. For example, in Figure 2, we create the tools **`seattle_sql`** and **`portland_sql`**, both of which use Google's `postgres-execute-sql` primitive to run SQL queries on the respective Postgres databases `seattle` and `portland`. We also define **`los_angeles_sql`**, which uses the MySQL primitive `mysql-execute-sql`, to query the `los_angeles` database.

Of course, there can be many such tool definitions for many different databases. Once the tools are defined, we can group them into flexible collections called *toolsets*. Each agent can then simply "subscribe" to the toolsets it needs.

As a simple example, suppose we want an agent to investigate crime statistics across cities in the "West Coast"—Seattle, Portland, and Los Angeles. To do that, it needs access to all three databases. All we have to do is bundle the corresponding tools from Figure 2 into a single *toolset*, **`west-coast-sql`**, and then subscribe the agent to it. It's just as easy to give the agent access to schema information: we simply subscribe it to **`west-coast-schema`**. The resulting configuration is shown in Figure 3.

```
tools:
  seattle_sql:
    kind: postgres-execute-sql # Google
    source: seattle # Postgres DB
  portland_sql:
    kind: postgres-execute-sql # Google
    source: portland # Postgres DB
  los_angeles_sql:
    kind: mysql-execute-sql # Google
    source: los_angeles # MySQL DB
  ...
```

Figure 2: A YAML fragment showing the configuration of database *tools*; schema-related tools are omitted for brevity.

In the same spirit, we might also maintain a **`washington-state`** toolset, bundling together the tools for databases from the Seattle area and other cities in WA. Within each database, we can import official data from various governmental sources, which THUCY can then explore when verifying claims about the state—exactly as in the ongoing investigation of the City Attorney's claim from Section 1.

If we later decide to remove access to a database (say, the Portland database), we only need to delete the corresponding tools in Figure 3—literally commenting out two lines of code from the configuration. Conversely, if we want to add another city into the mix, we simply append two more tools.

Agentic System

With the data layer now in place, we turn our attention to the core of THUCY: its multi-agent architecture. Connecting to databases modularly is only part of the challenge—the real difficulty lies in navigating and reasoning over them effectively. Our system tackles this through a team of three specialized expert-agents: the *Data Expert*, *Schema Expert* and *SQL Expert*. Each agent has a distinct role, specific instructions, clear output expectations, and subscribes to one of the

```yaml
toolsets:
  west-coast-sql:
    - seattle_sql
    - portland_sql
    - los_angeles_sql
  west-coast-schema:
    - seattle_schema
    - portland_schema
    - los_angeles_schema
  washington-state-schema:
    - seattle_sql
    ...
  washington-state-sql:
    - seattle_schema
    ...
```

Figure 3: Example YAML configuration of *toolsets*

two toolsets described earlier (**sql** or **schema**).

They are coordinated by the *Verifier*, a higher-level agent responsible for driving the verification process and producing the final verdict on claims—along with a transparent report containing explanatory SQL queries. Importantly, the three expert-agents are designed as *atomic* components: they never communicate directly with one another; they interact only with non-AI static tools exposed through their respective toolsets.

In this section, we discuss the rationale behind our design choices, the challenges we encountered, and the unique solutions that made our approach effective.

Data Expert Since the data environment is unknown, with potentially many databases and tables, we need a mechanism to rapidly survey the available landscape. This is the role of the *Data Expert*, which "subscribes" to the **schema** toolset. Its task is to perform a high-level scan of all accessible data sources and summarize what each source appears to contain.

The usefulness of this step might not be immediately apparent, but it is crucial: data exploration involves numerous tool calls and exposure to large amounts of low-level information—database, table, and column names; data types, schemas, and various metadata—that must be inevitably consumed to truly understand what the data is about. The *Data Expert*'s job is to "bite the bullet" navigating this chaos, and deliver a clean single-paragraph summary to the *Verifier*. This summary enables the *Verifier* to plan an effective verification strategy knowing the data sources it has in its disposal, while keeping its expensive context from being cluttered by useless details.

Schema Expert In order to write any successful query over relational databases, the first step is always to understand the schema. In theory, relational databases should have table and column names that are unambiguous, column types should match their intended semantics (e.g., an *age* column is a number and not text), and keys and foreign keys should be explicitly declared in the schema. In practice, this is rarely the case. Corporate or institutional databases have dozens or even hundreds of tables, each with dozens of attributes, and the table or column names are frequently non-descriptive. For example even the relatively well organized Crime Data for Seattle (Seattle Police Department 2025a) has opaque column names like `NIBRS Group AB` or `Beat`.

This is where the *Schema Expert* comes into play. Its high-level role is to answer arbitrary schema-related questions about the available databases. It is equipped with the **schema** toolset—similar to the *Data Expert*—which allows it to fetch detailed schema information from the connected databases. Unlike the *Data Expert*, however, it operates without guardrails and is in fact encouraged to dive deep into the structural details of the schemata. It can investigate nearly any aspect of the databases' design; from simple column names to specific constraints on those columns (e.g., foreign key relationships, nullability, and more).

However, misuse of its own tools can quickly clutter the agent's memory (for example, by the misfortune of querying the schema of a messy corporate database containing several hundred-column tables). To try avoid this as much as possible, we require one additional input to the *Schema Expert*. Along with the schema question, we must also provide a brief *context hint*—this is a short, high-level NL cue that steers the agent toward relevant databases (Figure 4).

All in all, the *Schema Expert* expects ① a NL schema question along with ② a *context hint*, and investigates the related data sources in order to provide a crisp, precise answer in NL. The exact response and format is left to the agent and depends on the question at hand. For example, a recent query was "*List all tables related to to crime, police incidents, offense categories, or year-by-year statistics.*" with the context hint of "*Seattle, WA*". The resulting answer was a well-structured, markdown-formatted summary detailing the relevant tables, column names, and types.

During execution, the *Verifier* frequently invokes the *Schema Expert* to answer different things about some database's schema (as in the example above). The responses are always clear, complete, and to the point—exactly what we want. This design "protects" the *Verifier*'s expensive context by allowing it to access highly curated schema information without having to endure the messiness of retrieving it.

SQL Expert Once a reasonable understanding of the database's schema has been established, the next step is to interact and play around with the data itself. For relational databases, this means writing SQL. The *NL to SQL* problem consists of automatically converting a natural language question over a relational database (with known schema) to an SQL query (Li and Jagadish 2014; Sen et al. 2020). Despite lots of progress in this space and the availability of popular benchmarks—like Spider (Yu et al. 2018), KaggleDBQA (Lee, Polozov, and Richardson 2021) and BIRD (Li et al. 2023)—NL2SQL remains a challenging problem for real-world scenarios, because of schema complexity, query ambiguity, and semantic mismatch (Floratou et al. 2024).

For example, suppose that we want to ask: *Which neighborhood of Seattle recorded the most parking tickets in the second quarter of 2025?* Everyone roughly understands what this question means, yet translating it into SQL quickly

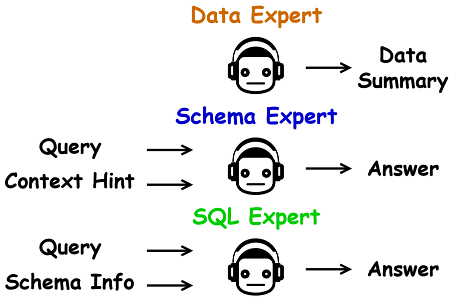

Figure 4: Inputs and outputs of the three expert-agents. The *Data Expert* is invoked without input and returns a concise high-level report of what the connected databases appear to contain. The *Schema Expert* expects a schema-related question along with a short hint about where to look (for example, "NYC database"), and returns a precise answer to that question. Lastly, the *SQL Expert* expects a question about the data along with specific information of the schema that is relevant to the question. The process usually involves a couple of SQL queries on the databases, depending on what the agent decides. At some point, it returns a clear answer together with the concrete SQL commands that verify it.

reveals several challenges. First, to compute total parking tickets, we must discover the semantics of "quarter" —is it relative to the calendar year or the fiscal year? Sometimes, the best we can do is to infer it by exploring the data itself. For example, by investigating the actual data, we might find out that the table conveniently includes concatenated information that indicate the quarter (e.g., "Q1" to "Q4"). Even if we are not that lucky, this process still helps us understand how dates are stored and organized. In other words, while the question is inherently ambiguous, careful inspection of the data often reveals its intended meaning. The first stage of query formulation consists of an interactive exploration of the data.

After this exploratory stage, the next stage is to write the query or queries that answer the question. Unfortunately, many things can still go wrong at this point. We might write a logically incorrect query (a semantic mismatch) and only realize it when no results appear—perhaps because we accidentally filtered everything out. On top of that, there are the usual syntax errors, some of which are due to the different vendor-specific versions of SQL (e.g., PostgreSQL, MySQL, Oracle, etc.) of the available databases.

In summary, NL2SQL is not a one-shot process, but requires a continuous back-and-forth with the database through SQL queries. In THUCY this is the role of the *SQL Expert*. This expert emulates the interactive process in order to answer questions over the available relational databases. To ensure transparency, we instruct it to always also return the concrete evidence that supports its answer. We want every result to be fully traceable back to the data through concrete SQL. We instruct the *SQL Expert* to exclude from the answer SQL queries that are irrelevant to the final answer (e.g., exploratory or failed queries).

The *SQL Expert* expects two inputs: ① the NL query itself, and ② the relevant database schema information (Figure 4). The latter contains all the necessary schema details the agent might need to write SQL queries within the scope of the question—such as the explicit names of the relevant databases, tables, relationships, and columns. This way, the agent can focus precisely on the relevant parts of the data, without being distracted by the rest of the environment. There might be countless other tables or databases available, but we want our *SQL Expert* to enter a kind of "tunnel-vision" mode, concentrating solely on the question at hand and on the small relevant portion of the data landscape that contains the answer.

Verifier Finally, the role of the *Verifier* is to coordinate all other agents, and verify the factual claim requested by the users. The *Verifier* produces two outputs: a verification verdict (one of *Verified*, *Partly Verified*, *Partly Inaccurate*, or *Inaccurate*), and an analytical report containing the SQL queries that explain and support this verdict. This report is organized in a clear, chronological way, effectively walking the user through each stage of the verification process. If desired, the data analyst can execute the explanation SQL queries herself, examine their outputs, modify and re-execute them, until she is completely satisfied with the veracity and generality of the claim.

To achieve this task, the *Verifier* interacts with all other agents in the system. In order to ask the *SQL Expert* something about the data, it will provide it with the relevant schema information indicating where exactly to look (Figure 4), obtained from the *Schema Expert*, while the latter requires information from *Data Expert*.

To summarize, the *Verifier* begins by asking the *Data Expert* to provide an overview of the available data sources. Using this information, the *Verifier* consults the *Schema Expert* to obtain the schema of the relevant databases, initially at some high level of detail (for example, only the table names without their attributes). Next, it invokes the *SQL Expert* to ask a question about a specific step of the claim verification, and obtains concrete, verifiable answers consisting of both SQL queries and their results. Usually, more information is needed, and the *Verifier* repeats this in a cycle: ask the *Schema Expert* for more detail, in order to ask the *SQL Expert* new queries; if the answers remain unsatisfactory, the *Verifier* may decide to ask the *Data Expert* for additional relevant data sources, and the process continues. Eventually, the *Verifier* is satisfied and writes the answer to the user.

Importantly, the *Verifier* never interacts directly with the messy data sources—it leaves the "dirty work" to the three experts. They are the ones who dive into the details, explore the data with whatever trials and tribulations, and handle its inevitable quirks. The *Verifier* simply asks the right questions and receives informative, concise, and crisp answers in return; without ever touching the chaos underneath. Thus, its context remains light, focused, and packed only with the most useful information. This efficiency allows us to equip the *Verifier* with a powerful model.

Orchestration

In practice, our three expert-agents are wrapped as callable functions and exposed to the *Verifier* as *tools*. This allows the *Verifier* to invoke any of them directly, much like calling a non-AI tool like a calculator. The architecture follows an “Agents as Tools” pattern, where specialized agents are encapsulated as a tools with clearly defined inputs and outputs.

For the expert-agents, we persist memory only within a single tool invocation, not across different calls. This design choice makes the agents reusable atomic components—any *lead* agent, such as the *Verifier*, can seamlessly employ them without inheriting messy context from previous runs. This keeps the multi-agent system simple, modular, and easy to extend.

3 Verification Example

Now, we can turn our focus to the verification example introduced in Section 1. As a reminder, Ann Davison, Seattle’s City Attorney, made the following statement in the 2024 Annual Report:

> *I am pleased to acknowledge that 2024 saw a reduction in property crime and violent crime in Seattle.*

We set out to verify this claim using THUCY. We have access to a PostgreSQL database with the City of Seattle’s official crime data. We had downloaded this data from the Seattle Police Department (2025b) in the form of a CSV file, then uploaded it into PostgreSQL. It contains roughly 1.5M rows of all recorded crimes from 2008 to the present. To verify claims like that by the city’s attorney, we invoked THUCY with the verbatim claim above. After a few minutes, our multi-agent system produces the following report and verdict (excerpted verbatim):

Findings:
- *Property crime increased from 40,951 (2023) to 41,220 (2024), a rise of 269 incidents (∼ 0.7%).*
- *Violent crime increased from 5,435 (2023) to 5,477 (2024), a rise of 42 incidents (∼ 0.8%).*

Conclusion: *The claim is contradicted by the data: both property crime and violent crime were slightly higher in 2024 than in 2023 in the Seattle dataset examined.*
Verdict: *Inaccurate*

An important feature of THUCY is that it also returns the explanatory SQL query (or queries) that lead it to its verdict. These queries can be directly inspected, executed, and reproduced by expert users. In our example, THUCY generated the query shown in Figure 5. In essence, the query groups crimes by year and category, and then counts the number of incidents—exactly what we would expect for this verification. The output of the query is also shown in the figure. It was easy to run this query ourselves and confirm the correctness of THUCY’s verdict; we show the answers in the figure. We also checked these results on the interactive crime dashboard of the City of Seattle (Seattle Police Department 2025a), and got the same results.[2]

```
SELECT
 EXTRACT(YEAR FROM offense_date)::int
 AS year,
 offense_category,
 COUNT(*) AS incident_count
FROM public.crime_data
WHERE offense_category IN
    ('PROPERTY CRIME','VIOLENT CRIME')
 AND offense_date >= '2023-01-01'::date
 AND offense_date < '2025-01-01'::date
GROUP BY 1, 2
ORDER BY 1, 2;
```

Year	Category	Incidents
2023	Property Crime	40,951
2023	Violent Crime	5,435
2024	Property Crime	41,220
2024	Violent Crime	5,477

Figure 5: SQL query and results produced by THUCY when verifying the City Attorney’s claim. The query groups crimes by year and category and counts total incidents.

4 Experiments

In this section, we present the experimental evaluation of THUCY. We first describe the widely used *fact verification* benchmark TabFact, followed by the baselines. Next, we outline the framework in which THUCY was built and the LLMs it uses. Finally, we present our findings, which show that THUCY decidedly surpasses the state of the art.

Benchmark We conduct experiments on TabFact (Chen et al. 2020), a widely used benchmark for fact verification over Wikipedia tables. The task is to determine whether a claim holds given the evidence in a relational table. The claim is labeled “false” if any part of it conflicts with the data from the table. Many cases involve subtle linguistic reasoning and common sense. Following all prior work (Nguyen et al. 2025; Zhu et al. 2024; Zhang et al. 2024), we evaluate on the small test split of TabFact, which contains roughly 2k fact-table pairs.

Baselines We compare against recent fact-verification systems that all rely on LLMs, as these have achieved state-of-the-art performance (Zhu et al. 2024). We do not reimplement the baselines; instead, we report the results provided in their original papers for the same task and dataset.

We compare against BINDER (Cheng et al. 2023), DATER (Ye et al. 2023), CoTable (Wang et al. 2024), ReActTable (Zhang et al. 2024), AutoTQA (Zhu et al. 2024), and POS (Nguyen et al. 2025).

Among them, AutoTQA is particularly relevant, as it also builds a multi-agent system and is the only one in the literature to also support cross-table querying. Their agents follow a cyclic orchestration pattern—executing, critiquing, and refining plans in a loop. Our approach differs in two main ways: ① THUCY is agnostic to the underlying data environment, and ② it provides concrete traceable evidence along-

[2] In the dashboard, make sure that “all” is selected in *Precinct*.

side the answers. We also take a different stance on agent orchestration: instead of cyclic pattern, we employ decoupled, specialized expert-agents. This choice is validated by recent successful applications in industry (Anthropic 2025).

POS is also related to our work, as it focuses on *interpretability*. It returns the execution plan to the user as a logical sequence of NL *atomic* steps. We differ in two key ways: ① we output concrete SQL queries, eliminating any suspicion of hallucinations, since expert users can directly verify them; and ② we are not constrained to an answer coming from a single query. In contrast to POS, which assumes the final answer is produced by a *single* SQL query, we allow—and in fact encourage—multi-step reasoning where potentially many arbitrary queries contribute to the final answer in different ways.

Setup We built THUCY using the OpenAI Agents SDK. Following our discussion in Section 2, we equip the *Verifier* with a highly capable model (GPT-5), since its context remains lightweight. We then experiment with the expert agents—*Data Expert*, *Schema Expert*, and *SQL Expert*—using two model variants: GPT-5-mini and GPT-4o-mini.

Results As we can see in Table 2, THUCY beats the previous state of the art by 5.6 percentage points, setting a new best-known result on TabFact at 94.3%. To test the robustness of THUCY, we also swapped the models of our three expert agents for GPT-4o-mini, aligning them to those used in the baseline systems (e.g., we match POS). The outcome remains the same: THUCY outperforms the previous state of the art by 5 points in accuracy. This result is especially encouraging—it shows that THUCY remains effective even when the individual agents use less capable models. It also reinforces our design choice of specialized, task-specific agents, since we can confidently downgrade their models to reduce cost without sacrificing much. We believe this decomposition of the overall task into smaller, well-defined subtasks, each handled by a dedicated expert agent under a single *lead* agent, plays a central role in these improvements.

5 A Journalistic Tale

A very recent article by MyNorthwest (2025) claimed that violent crime in *downtown* Seattle had "plummeted" during the summer months compared to the same period last year. The second sentence of the article reads:

> *Between June and August 2025, officials reported that violent crime incidents in downtown Seattle dropped by 36% compared to the same period in 2024.*

Within just a few hours, other outlets—including Kiro7 (2025) and Yahoo News (2025)—had picked up and republished the same story, all citing the original source.

Naturally, having built THUCY and with the crime dataset from the Seattle Police Department (2025b) already in hand, we were eager to see what it would say. We submitted the exact claim verbatim and waited a few minutes for the answer. Unlike the earlier statement by Seattle's City Attorney (Sections 1 and 3), however, this one was trickier. The term, "downtown", in particular, can mean different things depending on who you ask. After a few minutes, THUCY returned the following report (excerpted verbatim):

Conclusion

- *The Seattle crime data do not support the claim that "violent crime incidents in downtown Seattle dropped by 36%" in Summer (Jun–Aug) 2025 versus the same period in 2024. Depending on how "downtown" is defined, overall violent crime either increased modestly or decreased slightly, but nowhere near 36%.*

Verdict*: Inaccurate*

Assumptions and notes

- *"Downtown Seattle" was operationalized as the neighborhoods BELLTOWN, DOWNTOWN COMMERCIAL, and PIONEER SQUARE, and also tested with CHINATOWN/INTERNATIONAL DISTRICT included.*

What THUCY considered as "Downtown Seattle" is, in principle, reasonable. Still, before accepting the verdict, we wanted to dig deeper. The dataset includes a `neighborhood` attribute, which THUCY correctly[3] leveraged to filter by the relevant neighborhoods. This is exactly what a data analyst would do too. But this raised an interesting question: could there exist some other combination of neighborhoods—perhaps the one implicitly used by the news articles—for which the 36% drop actually holds?

We dug deeper into the SQL queries produced by THUCY. As experienced SQL users, we tweaked those queries to define "downtown" geographically instead: based on the distance from Seattle Central Library (which is undoubtedly *downtown*). To our surprise, when we restricted to crimes only within a radius of about 0.7km, the trend of the claim begun to emerge (Table 3). That only made us more determined to get to the bottom of this.

Table 2: Accuracy (↑) on the small test set of the TabFact Benchmark. Some papers decided to re-run the same experiments of previous methods using newer models, so we report the new results as well. Each entry points to its source paper.

Method	**Model**	**Acc (↑)**
BINDER (Cheng et al. 2023)	Codex	85.1%
BINDER (Nguyen et al. 2025)	GPT-4o-mini	84.6%
DATER (Ye et al. 2023)	Codex	85.6%
DATER (Nguyen et al. 2025)	GPT-4o-mini	81.0%
CoTable (Wang et al. 2024)	PaLM 2	86.6%
CoTable (Nguyen et al. 2025)	GPT-4o-mini	84.2%
ReAcTable (Zhu et al. 2024)	GPT-4	83.4%
ReAcTable (Zhu et al. 2024)	GPT-4-turbo	85.0%
ReAcTable (Zhang et al. 2024)	Codex	86.1%
AutoTQA (Zhu et al. 2024)	GPT-4	87.4%
AutoTQA (Zhu et al. 2024)	GPT-4-turbo	88.7%
POS (Nguyen et al. 2025)	GPT-4o-mini	82.7%
THUCY (ours)	GPT-4o-mini	93.7%
THUCY (ours)	GPT-5-mini	**94.3%**

[3] Or rather *incorrectly*, as we will see in Section 6

Table 3: Cumulative violent crime counts and percentage reduction (Jun–Aug 2024 vs. Jun–Aug 2025). Distance is counted from Seattle's Downtown Library using latitude and longitude coordinates available in the data.

Distance	2024	2025	Reduction
< 0.5km	30	37	-23.33%
< 0.7km	112	87	22.32
< 1.0km	178	146	17.98
< 1.5km	302	289	4.30

After further investigation on the Web, we finally uncovered the source of the confusion. The original news source came from a different article, published by the Downtown Seattle Association (2025), which stated:

> *Violent crime incidents in Seattle* ***police's M sectors*** *(the downtown core) declined 36% between June–August 2025 compared to the same period in 2024.*

This claim is far more specific: it reveals that the 36% refers specifically to the *police's M sectors*. Admittedly, we were not familiar with this terminology. So, once again, we invoked THUCY with the exact wording of this claim. This time, it returned the following report:

> ***Summary conclusion***
>
> – *Using report_datetime (report month), violent crime incidents in Seattle Police's sector M (downtown) fell from 105 in June–August 2024 to 67 in June–August 2025: a -36.19% change, which rounds to -36%. This matches the claim.*
>
> ***Verdict****: Verified*

With the extra *M sector* information at hand, THUCY was able to verify it. We noticed that this time THUCY took a different route: it filtered the data using attributes like `sector`. We also verified THUCY 's findings by cross-checking the results on the interactive crime dashboard of the Seattle Police Department (2025a). The numbers match perfectly.[4]

Even though we are not journalists, this whole process convinced us even further of the urgent need for journalistic tools that actually produce the concrete SQL evidence of their verdict. THUCY is one of them. It doesn't *just* give a verdict—the story doesn't *just* end there. It can be transformed, magnified, and turned to something greater. This is what Cohen et al. (2011) envisioned long ago in their pioneering seminal work on *computational journalism*.

6 Limitations & Future Work

Dirty Data. Coming back to the previous example, when we first gave THUCY the ambiguous claim about *downtown* Seattle, it made some assumptions about the `neighborhood`. It then filtered this attribute with the values it considered as *downtown*. So far so good—but what THUCY missed was that this column has many missing values. In fact, roughly 50% of them are missing. Since THUCY also returns the concrete SQL, we were able to spot this immediately.

Assumptions & Ambiguity. Another direction we want to explore is controlling how much the agents rely on assumptions. Assumptions are useful as this is the only way to combat *ambiguity* in both the data and user questions. However, they can also introduce subtle errors (for example, using the wrong current date). We want to experiment with ways to make these assumptions better grounded. One idea is to create another specialized expert-agent that searches the web.

Quantitative Evaluation. In the evaluation of THUCY, we used TabFact (Chen et al. 2020). There is, however, a mismatch: we propose a system that can navigate data environments with many databases and tables, while our evaluation is conducted on a single-table benchmark. This is indeed the case, but to the best of our knowledge there is no fact-verification benchmark in the literature that focuses on large-scale cross-table or cross-database data. We believe this is an important next step for fact-verification, and we are actively working on creating one.

Ablation Studies. As we were building THUCY, we manually tested and refined each agent, observing both their individual behavior and their interactions within the full system. However, in this work we do not present systematic ablation studies. A careful evaluation and a systematic study of the contribution of each component—both in isolation and by removing individual agents from the system—is warranted.

Stateless Expert-Agents. Each expert agent does not preserve memory across tasks. This is a deliberate design choice, as it allows THUCY to operate in dynamic data environments. However, this increases cost, since agents must re-discover information across tasks.

Expensive. Lastly, multi-agent systems like THUCY burn through tokens fast (Anthropic 2025). In our case, fact-checking all 4K examples in our experiments cost about $183.9 in total. That comes out to roughly 5¢ per example. In messy real-world fact-checking scenarios like the one discussed earlier (Section 5), the cost rises to 20¢ per verification. Still, we believe that our journalistic use case is high-stakes enough that this trade-off is worthwhile. After all, we can easily imagine journalists at the *New York Times* being more than happy to spend a few dollars to have their articles *stamped* by THUCY as *verified* and fault-proof.

7 Conclusion

We described our preliminary results for THUCY, the first multi-agent claim-verification system that operates over multiple relational databases and provides the concrete SQL evidence behind its verdicts. THUCY remains agnostic to the data environment prior to deployment and must therefore figure everything out from scratch. Our experimental results on a widely used fact-verification benchmark highlight the strength of our multi-agent design. THUCY improves the current state of the art in claim verification.

8 Acknowledgments

We thank the reviewers. This work was supported by NSF III 2507117, NSF SHF 2312195, and NSF IIS 2314527.

[4] When reproducing the results, after choosing the *year* and *offense category*, keep only *beats* M1–M3 selected; this is sector M.

References

Agarwal, A.; Yarnall, T.; Mauser, A.; Pimpalkhute, H.; Reini, J.; and Roy, R. 2025. Introducing Snowflake Managed MCP Servers for Secure, Governed Data Agents.

Anthropic. 2024. Introducing the Model Context Protocol.

Anthropic. 2025. How we built our multi-agent research system.

Buvaraghan, H.; and Egan, D. 2025. MCP Toolbox for Databases: Simplify AI Agent Access to Enterprise Data.

Chen, W.; Wang, H.; Chen, J.; Zhang, Y.; Wang, H.; Li, S.; Zhou, X.; and Wang, W. Y. 2020. TabFact: A Large-scale Dataset for Table-based Fact Verification. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net.

Cheng, Z.; Xie, T.; Shi, P.; Li, C.; Nadkarni, R.; Hu, Y.; Xiong, C.; Radev, D.; Ostendorf, M.; Zettlemoyer, L.; Smith, N. A.; and Yu, T. 2023. Binding Language Models in Symbolic Languages. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Cohen, S.; Li, C.; Yang, J.; and Yu, C. 2011. Computational Journalism: A Call to Arms to Database Researchers. In *Fifth Biennial Conference on Innovative Data Systems Research, CIDR 2011, Asilomar, CA, USA, January 9-12, 2011, Online Proceedings*, 148–151. www.cidrdb.org.

Downtown Seattle Association. 2025. Economic Revitalization - Tracking downtown revitalization.

Floratou, A.; Psallidas, F.; Zhao, F.; Deep, S.; Hagleither, G.; Tan, W.; Cahoon, J.; Alotaibi, R.; Henkel, J.; Singla, A.; Grootel, A. V.; Chow, B.; Deng, K.; Lin, K.; Campos, M.; Emani, K. V.; Pandit, V.; Shnayder, V.; Wang, W.; and Curino, C. 2024. NL2SQL is a solved problem... Not! In *14th Conference on Innovative Data Systems Research, CIDR 2024, Chaminade, HI, USA, January 14-17, 2024*. www.cidrdb.org.

Ganguly, R. 2025. Introducing the Azure MCP Server.

Gonzales, E.; and Murching, S. 2025. Announcing managed MCP servers with Unity Catalog and Mosaic AI Integration.

Kiro7. 2025. Violent crime plummets 36% in downtown Seattle, lowest since 2017.

Lee, C.; Polozov, O.; and Richardson, M. 2021. KaggleDBQA: Realistic Evaluation of Text-to-SQL Parsers. In Zong, C.; Xia, F.; Li, W.; and Navigli, R., eds., *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, ACL/IJCNLP 2021, (Volume 1: Long Papers), Virtual Event, August 1-6, 2021*, 2261–2273. Association for Computational Linguistics.

Li, F.; and Jagadish, H. V. 2014. NaLIR: an interactive natural language interface for querying relational databases. In Dyreson, C. E.; Li, F.; and Özsu, M. T., eds., *International Conference on Management of Data, SIGMOD 2014, Snowbird, UT, USA, June 22-27, 2014*, 709–712. ACM.

Li, J.; Hui, B.; Qu, G.; Yang, J.; Li, B.; Li, B.; Wang, B.; Qin, B.; Geng, R.; Huo, N.; Zhou, X.; Ma, C.; Li, G.; Chang, K. C.; Huang, F.; Cheng, R.; and Li, Y. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In Oh, A.; Naumann, T.; Globerson, A.; Saenko, K.; Hardt, M.; and Levine, S., eds., *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mehrotra, P. 2025. PayPal Begins Rollout of MCP Servers to Accelerate Agentic Commerce.

MyNorthwest. 2025. Violent crime plummets 36% in downtown Seattle, lowest since 2017.

Nguyen, G.; Brugere, I.; Sharma, S.; Kariyappa, S.; Nguyen, A. T.; and Lécué, F. 2025. Interpretable LLM-based Table Question Answering. *Trans. Mach. Learn. Res.*, 2025.

Seattle City Attorney's Office. 2024. 2024 Annual Report.

Seattle Police Department. 2025a. Seattle Crime Dashboard.

Seattle Police Department. 2025b. SPD Crime Data: 2008-Present.

Sen, J.; Lei, C.; Quamar, A.; Özcan, F.; Efthymiou, V.; Dalmia, A.; Stager, G.; Mittal, A. R.; Saha, D.; and Sankaranarayanan, K. 2020. ATHENA++: Natural Language Querying for Complex Nested SQL Queries. *Proc. VLDB Endow.*, 13(11): 2747–2759.

Wang, Z.; Zhang, H.; Li, C.; Eisenschlos, J. M.; Perot, V.; Wang, Z.; Miculicich, L.; Fujii, Y.; Shang, J.; Lee, C.; and Pfister, T. 2024. Chain-of-Table: Evolving Tables in the Reasoning Chain for Table Understanding. In *The Twelfth International Conference on Learning Representations, ICLR 2024, Vienna, Austria, May 7-11, 2024*. OpenReview.net.

Wikipedia. 2025. Thycidides — Wikipedia, The Free Encyclopedia.

Yahoo News. 2025. Violent crime plummets 36% in downtown Seattle, lowest since 2017.

Yao, S.; Zhao, J.; Yu, D.; Du, N.; Shafran, I.; Narasimhan, K. R.; and Cao, Y. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Ye, Y.; Hui, B.; Yang, M.; Li, B.; Huang, F.; and Li, Y. 2023. Large Language Models are Versatile Decomposers: Decomposing Evidence and Questions for Table-based Reasoning. In Chen, H.; Duh, W. E.; Huang, H.; Kato, M. P.; Mothe, J.; and Poblete, B., eds., *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR 2023, Taipei, Taiwan, July 23-27, 2023*, 174–184. ACM.

Yu, T.; Zhang, R.; Yang, K.; Yasunaga, M.; Wang, D.; Li, Z.; Ma, J.; Li, I.; Yao, Q.; Roman, S.; Zhang, Z.; and Radev, D. R. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In Riloff, E.; Chiang, D.; Hockenmaier, J.; and Tsujii, J., eds., *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, 3911–3921. Association for Computational Linguistics.

Zhang, Y.; Henkel, J.; Floratou, A.; Cahoon, J.; Deep, S.; and Patel, J. M. 2024. ReAcTable: Enhancing ReAct for Table Question Answering. *Proc. VLDB Endow.*, 17(8): 1981–1994.

Zhu, J.; Cai, P.; Xu, K.; Li, L.; Sun, Y.; Zhou, S.; Su, H.; Tang, L.; and Liu, Q. 2024. AutoTQA: Towards Autonomous Tabular Question Answering through Multi-Agent Large Language Models. *Proc. VLDB Endow.*, 17(12): 3920–3933.