AI Agent for User Data Analysis in Quality Assurance for Mobile Applications

André Kanakis
an8615ka-s@student.lu.se

Frida Engvall
fr4156en-s@student.lu.se

Department of Electrical and Information Technology
Lund University

Supervisor: Christin Swahn

Examiner: Christian Nyberg

December 19, 2025

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Writing SQL to extract usage data is a strenuous task that requires extensive knowledge of logging practices and user interactions within an application. For engineers and developers who write SQL only intermittently, the resulting context switching and re-familiarization can consume valuable work hours.

To address this challenge, an internal AI agent was developed to transform natural language into project specific SQL queries for BigQuery databases, reducing or eliminating the need for manual query writing. This automation allows engineers to focus on analyzing user interactions and deriving insights.

By analyzing the application source code and interviewing engineers of varying expertise, obstacles to efficient data-driven decision-making were identified. The complex table structure in BigQuery and the lack of detailed documentation on logging behavior were two major obstacles to efficient query writing that could benefit from AI assistance.

The result is a prototype of an AI agent, using the AutoGen framework, available through both terminal and visual interfaces that can be configured for different teams and projects, facilitating easier and faster data analysis. The agent incorporates domain- and project specific knowledge of what is logged and why it is logged through the use of descriptors in a custom tool to view the database schema.

Testing showed that six out of eight existing queries written by experienced engineers could be reproduced using natural language prompts in less than five attempts, with an average execution time of just over half a minute from submission to data retrieval. These results highlight the potential of using AI assisted query generation in quality assurance work for the development of mobile applications.

Keywords: Agent, AI, AutoGen, BigQuery, Data analysis, LLM, Mobile Applications, NLP, QA, Quality assurance, SQL

Sammanfattning

Att skriva SQL-satser för att hämta användardata är ofta en ansträngande uppgift som kräver god kännedom om hur datainsamlingen fungerar, och hur användare interagerar med applikationen. För utvecklare som endast skriver SQL ibland innebär det mycket kontextväxling och tid som går åt till att åter bekanta sig med dataanalyseringsarbetet.

För att lösa detta problemet utvecklades en intern AI-agent som omvandlar naturligt språk till projektspecifika SQL-satser för BigQuery databser. Därmed reduceras eller elimineras behovet av att manuellt skriva SQL-satser. Denna automation låter istället utvecklare fokusera på att analysera och dra slutsatser från resultatet gällande användarbeteenden.

Genom att analysera applikationens källkod och intervjua utvecklare med diverse erfarenhetsnivåer, kunde flera hinder för effektiva datadrivna beslut identifieras. Den komplexa databasstrukturen och avsaknad av detaljerad dokumentation över datainsamlingen var båda två tydliga hinder som stoppade utvecklare från att effektivt skriva SQL-satser, vilket kan mitigeras genom AI-assistans.

Resultatet blev en prototyp av en AI-agent, utvecklad med AutoGen-ramverket, som är tillgänglig både via text- och grafiska gränssnitt. Denna prototyp kan anpassas och konfigureras för nyttjade i olika teams och projekt, och därmed gynna för en enklare och snabbare dataanlys. Prototypen har projekt- och domänspecifik kunskap för data som samlas in genom inbyggd dokumentation i agentens *tool*, vilket används för att agenten ska se strukturen över databasen.

Tester visade att sex av åtta tidigare skrivna SQL-satser kunde återskapas med naturligt språk under fem försök, med en genomsnittlig exekveringstid på drygt trettio sekunder från start till hämtat dataresultat. Resultaten visar att AI-assisterad SQL-generering potentiellt kan effektivisera kvalitetsarbetet inom utveckling av mobilapplikationer.

Nyckelord: Agent, AI, AutoGen, BigQuery, Dataanalys, Kvalitetsarbete, LLM, Mobilapplikationer, NLP, QA, SQL

Preface

We would like to express our gratitude to our company supervisors, Anastasia Kantor and Oscar Porenius, for guiding us throughout our thesis, providing feedback, and sharing their expertise. Their support and motivation made this thesis a smooth and rewarding experience.

We also wish to thank the Mobile Apps department and the ACS Edge, Cloud & Mobile Apps Test Team at Axis for welcoming us warmly, engaging with our work, and generously offering their time for interviews and discussions.

Finally, we would like to thank our academic supervisor, Christin Swahn, and our examiner, Christian Nyberg, for overseeing this thesis and for their valuable feedback on the report and initial descriptions. Their guidance helped us maintain a steady pace while staying on track.

André Kanakis & Frida Engvall

Abbreviations

AI	Artificial Intelligence
ANSI	American National Standards Institute
API	Application Programming Interface
CLI	Command Line Interface
CSV	Comma-Seperated Values
FLOPS	Floating point operations per second
GenAI	Generative AI
IoT	Internet of Things
JSON	JavaScript Object Notation
LLM	Large Language Model
MCP	Model Context Protocol
NLP	Natural Language Processing
QA	Quality Assurance
ReAct	Reason + Act
SDK	Software Development Kit
SLM	Small Language Model
SQL	Structured Query Language
TUI	Text-based User Interface
VMS	Video Management System
YAML	YAML Ain’t Markup Language

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis was completed in collaboration with Axis Communications AB, specifically the Mobile Apps department working in End-to-End Solutions. In this chapter, an introduction of the company, the goals and the reason for this thesis will be presented.

1.1 Background

Axis is an industry-leading company that specializes in video surveillance. Their headquarters is located in Lund, Sweden, with several offices near the main office [1]. In total, Axis has more than 5000 employees [2] in more than 50 countries [1]. Axis was founded on the idea of connecting devices using networks in a technology called *ThinServer Technology*, which is today known as the Internet of Things *(IoT)* today [3].

The Mobile Apps department currently gathers user data when users interact with their mobile applications. They gather this data to improve the experience of end users [4]. Examples of what kind of user data is gathered for the mobile application *AXIS Camera Station* include: user interactions with the UI, communication between user and server, configuration of client hardware [5]. The user data collected cannot be linked to a specific user [6].

With more than 200 customer stories [7] and some customers using Axis products on 61 sites [8] and installing over 80 000 cameras [9], a substantial amount of user data is collected each day. Currently, the Mobile Apps team cannot feasibly analyze all of this data. Thus, Axis is interested in solutions to assist their developers and QA engineers with analyzing data. To achieve this, Axis proposes to develop an AI agent prototype that can complete some of the engineers tasks for them, such as creating and executing database queries with Natural Language Processing *(NLP)*.

An AI agent is created when a Large Language Model *(LLM)* is capable of making tool calls, allowing it to invoke and use functions of other programs to complete tasks autonomously [10]. Through effective use of system- and role prompting, instructing the LLM in how to behave and act, the agent can be configured to act like a QA engineer and obtain the relevant context needed for the tasks that the agent is expected to solve [11]. To help ensure the privacy of the collected user data, Axis intends to use a local LLM for the prototype.

1.2 Purpose

The purpose of this thesis is to investigate how a local LLM can be used as an AI agent to enhance the efficiency of user data analysis, and to develop a prototype that utilizes a local LLM to assist with user data analysis.

1.3 Objective

The objectives of this thesis are as follows.

- Develop a prototype that uses an AI agent to assist in user data analysis.
- Evaluate a prototype AI agent based on impressions and expectations from QA engineers on the Axis Mobile Apps team.
- Investigate potential implementations of an AI agent in a user data analysis-centric workflow.

1.4 Research problem

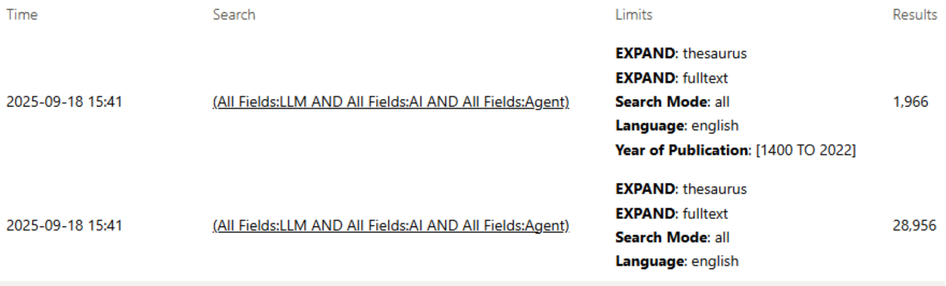
The thesis aims to address the following research problems:

1. How does Axis currently analyze user data for its mobile applications?
2. What are some current constraints limiting user data analysis?
3. Where can an AI agent be implemented within a user data analysis-centric workflow?
4. How can a prototype AI agent be implemented?
5. How can an AI agent visualize data and present results?
6. How can a prototype AI agent be evaluated?

1.5 Thesis motivation

AI agents are a new field of study, with more than 90% of the results from Lund University's library search tool, Finn, having been authored since 2022, as seen in figure 1.1. This new development is thought to mark a paradigm shift in how tech companies operate. Leaders in the tech industry are investing in AI agents and project that the market surrounding AI agents will continue to develop in the coming years [12].

With the advances of AI agents, socio-economic consequences will also follow. Agents will be able to enhance and replace human jobs. If agents are used to replace jobs currently performed by humans, mass unemployment and an increase in economic inequality may result. If agents are instead used to improve jobs, workers will have to adapt to new tasks. Although some jobs will disappear due to increased efficiency, new jobs will also emerge [13].



Time	Search	Limits	Results
2025-09-18 15:41	(All Fields:LLM AND All Fields:AI AND All Fields:Agent)	**EXPAND**: thesaurus **EXPAND**: fulltext **Search Mode**: all **Language**: english **Year of Publication**: [1400 TO 2022]	1,966
2025-09-18 15:41	(All Fields:LLM AND All Fields:AI AND All Fields:Agent)	**EXPAND**: thesaurus **EXPAND**: fulltext **Search Mode**: all **Language**: english	28,956

Figure 1.1: Finn search results for terms LLM and AI and Agent between different date ranges.

Several large AI companies present themselves as carbon neutral by purchasing carbon emission credits. The impact of AI on carbon emissions and energy use is significant and is likely to increase unless efforts are made. In 2019, training a NLP AI model required the equivalent of 300 000 kg of CO_2 emissions. In addition to using clean energy and more efficient components, the model size can be configured; using smaller models would reduce the amount of data needed during training and consume less energy [14].

We believe that experience with developing AI agents will be sought after in the coming years. Getting hands-on experience with this while being able to work with real data and stakeholders that can benefit from our work is an opportunity that we are very excited about. We also look forward to exploring this technology while taking into account the social, economical, and environmental consequences while working according to the Engineers of Sweden's code of honor [15].

1.6 Limitations

The scope of the thesis will be limited to Axis Mobile Apps products and the team's interests in collected user data. Only a selection of their products will be considered in this thesis. Only local LLM models will be used for the prototype.

1.7 Division of labor

Throughout the thesis, collaboration between authors was a focal point. We made sure to keep each other in the loop and receive feedback and confirmation from each other. An estimation of the division of labor can be seen in table 1.1.

Table 1.1: Division of labor between authors for thesis work

Task	**André Kanakis**	**Frida Engvall**
Literature study	40%	60%
Interview	30%	70%
Agent framework	80%	20%
Tool functionality	50%	50%
Database extraction	20%	80%
User interface	70%	30%
Report writing	50%	50%
Poster	60%	40%

Chapter 2

Technical background

This chapter explores some of the technologies, tools, and concepts that were used to complete this thesis.

2.1 Artificial Intelligence

Artificial intelligence is the imitation of human capabilities, such as learning, reasoning, and creativity [16]. The field of AI derived further concepts such as machine learning, deep learning, and generative AI. Generative AI *(GenAI)*, is a form of AI that can create seemingly original content such as images and texts [17].

2.1.1 Machine Learning

Machine learning is a concept where models are created by making decisions based on data and algorithms. A simple form of machine learning is *supervised learning*, by having training data that have been labeled by another entity, usually human operators, the model can learn to handle previously unseen data and apply patterns from the training data to make predictions based on the labels and the corresponding data [17].

2.1.2 Deep Learning

Using several layers of neural networks, a machine learning technique, a better representation of the human brain can be constructed. Using these layers, *unsupervised learning* is possible, allowing algorithms to group associations between data entries. This means that human intervention and large fabricated data sets are no longer required to train models [18], [17].

2.1.3 Generative AI

GenAI are deep learning models that are capable of creating new work that is based on and derivative of training data. By letting a deep learning model, called a *foundation model*, train on terabytes and petabytes of raw data before stepping into the next step; tuning. When tuning a model, the labeled data specific to the application domain of the model can be used to further fine-tune and train the model to work well for specific tasks [17].

2.2 Natural Language Processing

NLP is a field of machine learning that has long been used to analyze text. It can, for example, be used to gather keywords from input, guess sentiment, or be implemented in tools for automatic correction of spelling and grammar assistance of text [19].

NLP first preprocesses the input text, usually done by removing punctuation, words like "the" and "is", changing all letters to lowercase, and changing all verbs to their plain form. The preprocessing also tokenizes the input, splitting it into letters, n-grams, words, or sentences. Once the input has been preprocessed, it is ready for analysis [20].

2.3 Large Language Model

Large language models are foundation models focused on text generation and generative AI [17]. They are created to handle a variety of tasks. The training of these models is performed using *self-supervised learning*, a form of unsupervised learning. After training on trillions of sentences, an LLM can predict words and complete or compose new sentences [21]. Querying an LLM to complete a task is called *prompting*. Taking the context of the user's prompt and the conversation thus far, the LLM predicts the most likely next words based on its training data; this is the output that is generated. The output can be configured in several parameters, output length; to stop the generation of tokens *(words)*, temperature; which controls how random the output selection is, top K; varies the selection from the top K most likely output tokens, top-p; varies the selection of top tokens that are under the cumulative probability P. By changing these values, an LLM can be modified from generating the same response from the same prompt to generating varied and seemingly random responses [11].

2.3.1 System prompting

System prompts are used to help format the output to fit the requirements of the task at hand and to create an environment in which the LLM can act in a specific way. A system prompt can instruct the LLM to always output its answer in JSON, to answer very concisely, or to respond like it would believe a personal tutor would based on the models training data. When the LLM is first prompted, a system prompt is usually prefixed to the initial context window, shaping the conversation [11]. An example of system prompting can be seen in appendix B, figure B.1

2.3.2 Role prompting

Role prompts are a type of system prompt that instruct the LLM to act as a specific character, thus modifying the output to resemble how that character is represented in the training data. It is a small addition to a prompt that can change the tone, reasoning, and style of the output [11]. An example can be seen in appendix B, figure B.2

2.3.3 Contextual prompting

Contextual prompts are a form of system prompt that add information and details needed to complete the assigned task. A contextual prompt will guide the LLM to a more correct and effective output as the model will not make as many assumptions [11]. An example can be seen in appendix B, figure B.3

2.4 Agents

An agent is a system that can complete tasks and assignments on the user's behalf without the intervention of the user [10]. AI agents are agents that can reason, understand instructions in several forms, and complete tasks in a way that resembles how humans would approach a problem. An AI agent should function autonomously, being able to make sophisticated decisions and evaluate different resolution paths [22, pp. 2–7].

AI agents are powered by the use of LLMs, using models to process the natural language of the user's prompt and to generate a response [23]. AI agents can use LLMs to implement *ReAct*, a popular framework based on *re*asoning and *act*ing, where the agent first reasons about the tasks at hand and creates plan of action [24], [25].

2.4.1 AI Agent and Agentic AI

Agents using LLMs can be divided into AI agents and agentic AI. With AI agents being mostly singular agents that are used for modular, tool-assisted tasks and agenic AI consisting of large multi-agent orchestrated systems. This difference in scope means that the systems are better suited for different assignments and tasks, but also that their limitations and problems differ. Singular AI agents can struggle with hallucinations, limited planning, and limited autonomy with task variety. Agentic AI systems struggle with inter-agent errors, wrong context propagating between agents, increased unpredictability due to coordination breakdowns, and being a more complex system [26].

AI agents are generally narrow, they work in four steps; perception, reasoning, action, and learning. The agent perceives the task and reasons, creates a plan, before taking action, completes the assignment, then learns, adding to the context window, and expanding it's current knowledge as the context grows. These four steps can be modified with prompt engineering, adding system- or role-prompts to guide the agent into specific courses of reasoning or action [26]. This self-observing reasoning and action approach is able to outperform in certain tasks compared to a external feedback approach that is solely based on the current state of the environment, similar to what an agent might feed into another agent when not using a framework like ReAct [25]. Multi-agent systems, agentic AI, orchestrates several agents. This orchestration allows for agents to hold specific jobs and specializations, one agent can be in charge of planning, another agent in charge of formatting output. Agents in the agentic AI system need to be able to communicate and share context via communication channels such as memory queues or shared memory. This connection between multiple agents enables agentic AI systems to autonomously set goals, coordinate between agents, share context, and maintain a larger context buffer between tasks [26].

Since most agent tasks are narrow, using an LLM to orchestrate multiple agents that are powered by small language models *(SLM)* and that are specialized on specific tasks compared to generalized large language models can be more efficient. For starters, the cost of using a 7bn parameter SLM is 10-30 times cheaper compared to running a 70-175bn parameter LLM when taking latency, energy consumption, and FLOPS into account. SLMs also enable faster iteration when fine-tuning, allowing for companies to be more efficient and cost-effective when creating agents tailored to their specific workflow [27].

2.4.2 Tool calls

In order to increase the range of actions that an AI agent can take and to gather new information that was not in the models training data, agents use *tool calls*, also called *function calls*, to interact with other systems and APIs. When the agent recognizes that a tool is required, it checks its available tools and selects the tool that is best suited for the task [28]. The agent prepares the tool call with any necessary arguments, usually in a JSON schema, before sending the request to the tool API and executing a tool call. The agent then takes the response from the tool call and processes it, extracting the response, possibly reflecting on it, and generating a response [29]. A tool call generally consists of three components in JSON; a name to uniquely identify the tool, a description to inform the model of what the tool does, and parameters that inform the model of what input is required [30]. A definition of a simple tool can be seen figure 2.1, and the response that the model generates to use the tool can be seen in figure 2.2.

```
{
  "name": "write_file",
  "description": "Writes a file at the specified path",
  "parameters": {
    "filepath": "The path where the file is to be saved"
  }
}
```

Figure 2.1: A simple tool call definition in JSON that creates a file at a specified path.

```
{
  "tool": "write_file",
  "parameters": {
    "filepath": "~/.config/nvim/init.lua"
  }
}
```

Figure 2.2: A tool call of a function in JSON, listing the tool called potential parameter values.

2.4.3 Model Context Protocol

Model Context Protocol *(MCP)* is a standard protocol for connecting AI agents with data sources and tools, released by Anthropic in late 2024 [31]. MCP speeds up development and improves maintainability as developers no longer need to build custom connectors between each individual AI model and external service [32].

The architecture of MCP is divided up into three main components, the MCP host, MCP clients, and MCP servers. The MCP host is the application that the user interacts with, it can be a text editor or an AI model. It is the host that creates and manages MCP clients, instanstiating a client for each connection to a server, providing a 1:1 relationship between MCP clients and MCP servers. The MCP client manages the connection between the host and the service provided by the MCP server. An MCP server is the service that provides context back to the client [33]. Although it is called a server, it can be hosted locally on the device and externally on other devices, an MCP server is simply a program exposing functionality through a standard protocol [34]. An example of the interaction flow between a user and MCP servers can be seen in figure 2.3, demonstrating how an MCP client is part of the MCP host and is responsible for the connection with an MCP server.

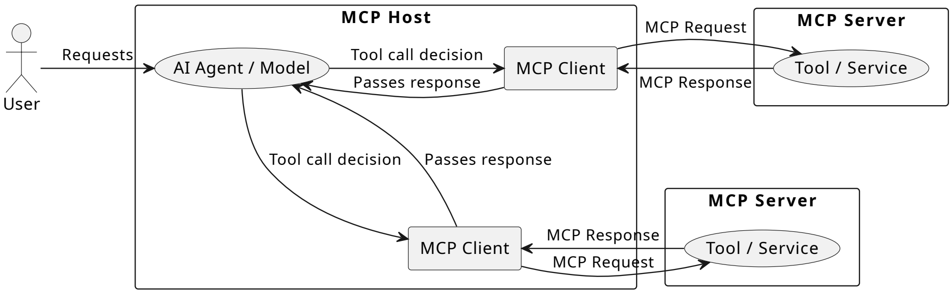


Figure 2.3: Interaction flow of Model Context Protocol between a user and external MCP servers.

2.5 AutoGen

AutoGen is an open-source framework by Microsoft for building AI agents and multi-agent systems by asynchronous messaging between agents. AutoGen focuses on being an easy-to-use framework that is suitable for quick development and research with *AutoGen-AgentChat* while also providing *AutoGen-Core*, a form of AutoGen with less scaffolding, but giving developers more control [35]. AutoGen has frameworks for prototyping without writing code, for conversational single- or multi-agent systems, and extensions for external tool calls and MCP connections [36].

AutoGen supports a paradigm called *conversation programming*, in which agents engage in discussions and converse with each other, utilizing the strength of the language models that power the agents. Conversation programming is divided into *computation*, where the agents compute their message to send in the conversation, and *control flow*, controlling the order of messages and conditions to let the conversation flow [37]. In figure 2.4, an example of a conversation between a CoderAgent that can write and execute code a ResearchAgent with internet search tools available, and a ReviewerAgent that reviews code. The UserProxyAgent allows human-in-the-loop execution where a user can provide feedback to the team of agents and interact with them [38].

AutoGen enables several types of conversation for the agents. They can be structured in a BaseGroupChat; where all participants share their context to all other participants when publishing a message, SelectorGroupChat; where a ChatCompletion-model selects the next speaker depending on the previously sent message and messages are sent to all participants, RoundRobinGroupChat; where participants take turns in sending a message to all participants, Swarm; where the current speaker can select who should speak next with a HandoffMessage, and MagenticOneGroupChat; where a MagenticOneOrchestrator-agent controls the flow of conversation in an attempt to complete the task efficiently [39].

2.6 BigQuery

BigQuery is a managed, serverless, data warehouse offered by Google. Providing scalability and flexibility to storage solutions and including several tools for data insight and analytics [40]. Consisting of two parts, a storage layer that stores and optimizes the data and a compute layer to provide analytics. The separation of storage and compute allows both parts to work independently without affecting the performance of the other [41].

Several APIs are provided for interacting with datasets. The *Python Client Library* can be used to connect to BigQuery and execute queries in Python programs [42]. Google also provides an agent development kit, a framework for developing and running AI agents with first-party tools for BigQuery [43], and an MCP server for integration with BigQuery [44].

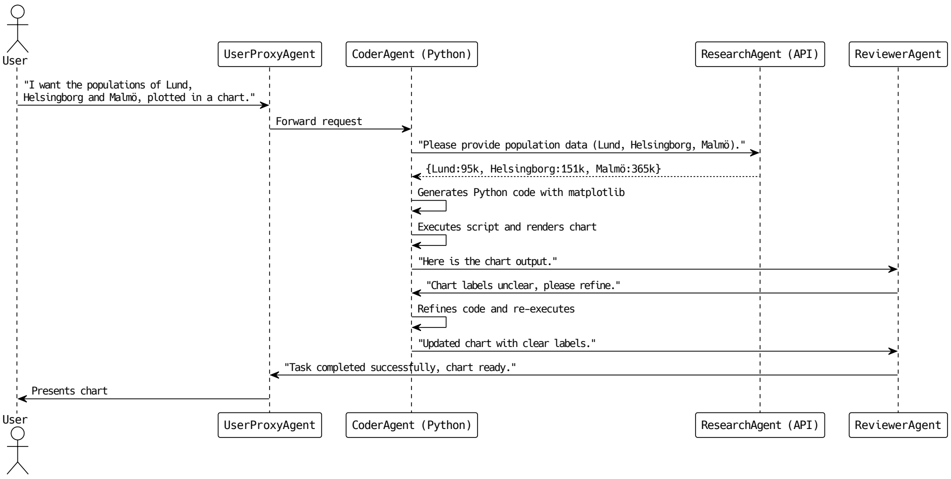


Figure 2.4: Sequence diagram showcasing how AutoGen agents can work in a team to complete the user's task of producing a graph over population for some Swedish cities.

2.6.1 GoogleSQL

GoogleSQL is an American National Standards Institute-compliant *(ANSI)* SQL-dialect that is used in Google's BigQuery. This dialect contains some new and unique features of SQL to use when querying. It includes a pipe syntax, similar to Elixir [45], that can be used to manipulate and send the result into another pipe function. Queries can also start with the `FROM`-clause when using pipe syntax [46]. A comparison of what can be done using the pipe operator can be seen in figures 2.5 and 2.6. Pipe syntax provides an alternative order of operations and makes it easier to add more operations to a pre-defined query. GoogleSQL also supports nested records and unnesting elements into one row each [47].

2.7 Looker Studio

Looker Studio provides visualization of data through customizable dashboards. Dashboards and reports can be connected to BigQuery for a complete ecosystem from user analytics, to storage, to data visualization [48]. Reports can be generated with the drag-and-drop visual feature, but also with custom SQL queries using GoogleSQL. Scheduled BigQuery jobs can be integrated to automatically update Looker Studio dashboards [49].

```
FROM dealership.Sales
|> WHERE sales > 0
|> AGGREGATE SUM(price) as total_sales, COUNT(*) as num_sales
    GROUP BY salesperson
|> JOIN dealership.Staff USING (salesperson)
```

Figure 2.5: A query for generating salesman statistics for a car dealership, using GoogleSQL pipe syntax for a more terse query.

```
SELECT
  st.name AS salesperson_name,
  s.salesperson,
  SUM(s.price) AS total_sales,
  COUNT(*) AS num_sales
FROM dealership.Sales AS s
JOIN dealership.Staff AS st
  USING (salesperson)
WHERE s.sales > 0
GROUP BY salesperson, salesperson_name;
```

Figure 2.6: A query in traditional SQL for generating salesman statistics for a car dealership.

Chapter 3

Methodology

In this chapter, the different phases of the thesis will be presented. Details about how employees' interviews were conducted, how communication between different parties was handled, and how different techniques and technologies were tested and implemented in the prototype.

3.1 Phases of work

The thesis was divided into three parts during planning. Each part being 5 weeks, for a total of 15 weeks, matching a pace of full-time studies for 15 weeks at 1,5 HP per week as outlined for the course. The initial GANTT chart can be seen in figure 3.1. The GANTT chart is divided into three main parts, phase 1: discovery phase between weeks 36 - 40, phase 2: implementation phase between weeks 40 - 45, and phase 3: evaluation phase between weeks 46 - 50. Some work between phases flowed into each other, though they largely focused on a distinct part of the thesis work.

3.1.1 Discovery phase

The first phase focused on exploring the area for the thesis work and acquiring relevant knowledge. This included a literature study, library databases were used to find scientific papers about relevant work that has already been conducted within the topic. In addition, different guides and documentations from the internet that might be relevant for the thesis work were read. Several interviews with employees were also conducted to get a better understanding of the company, current work processes, and the products that were in focus.

3.1.2 Implementation phase

For the second phase, the implementation and development of the prototype was the main focus. Frameworks and libraries were tested and ultimately selected. Small programs focused on each part of the prototype were developed, and quick iterations were made to develop the prototype as efficiently as possible. Prompt engineering and testing were divided into two parts and completed after the agent

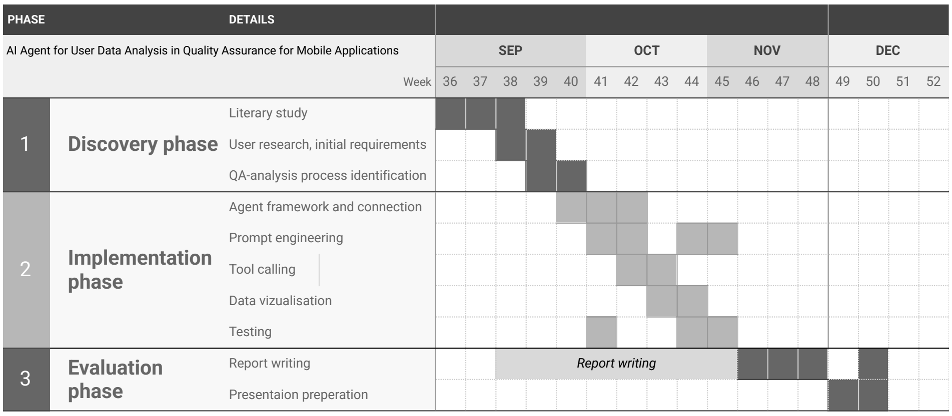


Figure 3.1: Initial GANTT chart over the thesis work.

framework and model connection. After the tooling capability was completed, more tests and refinements of prompt engineering were conducted.

3.1.3 Evaluation phase

The last phase of the thesis was the evaluation phase, this focused on analyzing and discussing our results and report writing. The report was continuously written throughout the period, but in this last phase, time was dedicated exclusively to work on the report and presentation. These weeks would also work as a buffer to add more time to the implementation phase if necessary.

[!h]

3.2 Communication

Effective communication was essential to ensure smooth development cycles and to maintain the expected level of quality throughout the thesis work. Communication took place on several levels; between the authors themselves, with the company and its employees, and with the academic supervisor. Each of these channels served different purposes and contributed in distinct ways to the progress of the thesis.

3.2.1 Between authors

Both authors were seated in the office during the day, sharing a room to work on the thesis. This meant that communication and collaboration flowed smoothly. It was easy to ask for feedback and to discuss ideas. Outside of the office, communication was conducted through asynchronous messaging services.

3.2.2 With the company and company supervisors

As the authors were in the office, it was easy to communicate with the company supervisors and coworkers. This meant that interviews could be conducted in person and that spontaneous interactions could regularly occur. Communication with team and department members was also conducted via asynchronous communication such as email and Microsoft Teams. The two supervisors held an informal meeting with the authors each morning, where the authors could discuss the planned tasks for the day and get quick feedback from the supervisors.

3.2.3 With the academic supervisor

Communication with the supervisor was maintained through bi-weekly digital meetings complemented by email correspondence. These meetings served not only to provide structured feedback on the report, but also to ensure that the work progressed according to the planned GANTT-chart (figure 3.1). In addition, the supervisors input helped resolve uncertainties about research design and adjustments when goals or timelines changed.

3.3 Elicitation

This work used elicitation techniques, not to formulate structured requirements, but rather to identify the needs of engineers, how an AI agent could assist them, and to learn about how the team analyzes data.

3.3.1 Literary study

In the first phase, literary studies were conducted to gather information and gain knowledge about the emerging field of LLMs powered AI agents. LUBSearch and FINN were the main sources of literature. Websites from industry leaders were also used whenever a focused explanation of a term of a technology was required. Sources were downloaded locally in a pdf-format whenever possible and annotated by the reader before being uploaded to a shared network-folder where a back-up would be stored in case the original host was shut down or the file were to be removed.

Whenever a source that seemed relevant was found, it was added to a spreadsheet, marked once it was read, graded according to how relevant it felt, and notes about the source were documented. With this spreadsheet, differing interpretations and opinions could quickly be spotted, and problematic sources were quickly filtered out in a way that avoided both authors having to read the same sources. More functionality was added to the spreadsheet as work on the report began, such as marking which sources had been added to the list of references.

3.3.2 Interviews

The interviews were held in the first phase to learn how different teams in the Mobile Apps-department work with data and data analysis, and to focus on where a solution could assist them in their work. A total of 6 different teams were interviewed, with some teams having more than one representative being interviewed. The bookings were made through a booking page where coworkers could pick a date and time that suited them from a list of available slots.

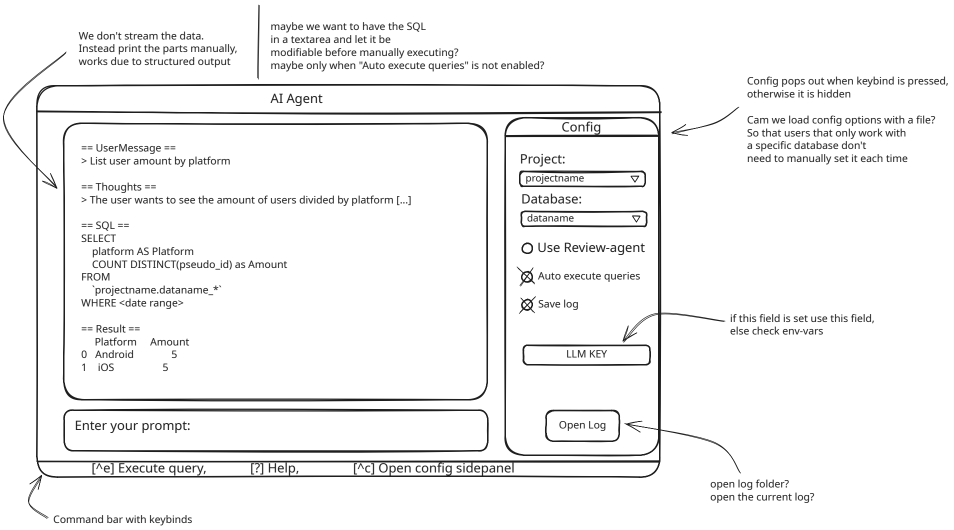
A total of 12 interviews were conducted over the course of one week, each scheduled for 30 minutes. The interviews held on the first day were used to test the questions and get acquainted with the interview process. The interviews were in a semi-structured format where the questions were quite open-ended and allowed for a wide range of answers. Some questions were marked optional by the interviewers as potential follow-up questions. After adjustment from the first day, a total of 11 questions were used in the interviews. The questions were divided up into 3 parts; first, A-labeled questions about the assignments of the interviewee and how they work with data analysis, then B-labeled questions about using and working with BigQuery, and finally, C-labeled questions about current and future AI usage. The nterview questions are listed in appendix A. The interviewees came from a mixed background, working within different teams in the Mobile Apps department, ranging from developers to managers. Some interviewees did not work with BigQuery, and thus, part of their interviews were omitted. The interview questions were based on the research problems, general curiosity and reworked according to the Mom-test [50].

After each interview, a 15 minute recess was held where the interviewers could adjust and review the notes that were taken during the interview. One interviewer led the interview and asked the prepared question, while the other took notes. If the note-taker ever felt like asking a follow-up question or veering off course, the lead interviewer would take over the responsibility of taking notes until the interview returned to the line of questioning. After all interviews were held, they were summarized in a managable format through generalized answers for the entire group of interviewees.

3.3.3 Prototyping

As the thesis focused on exploratory work and was conducted as an internal project with little or no formal requirements [51, p. 8], prototyping was selected as a form of investigation and development. With prototyping, a test run and validation of the usage of AI agents for current data analysis tasks and the results of the prototype can form design and product-level requirements to be used for further development [51, pp. 344–345]. By prototyping, costs can be kept low and requirements can be naturally identified and implemented as the prototype moves through its iterative process [52, pp. 17–19].

The user interface of the prototype was sketched out with some thoughts and ideas, the initial sketch can be seen in figure 3.2. This initial version was later modified, and more details and features were added as the prototype evolved and more functionality emerged.



We don't stream the data.
Instead print the parts manually,
works due to structured output

maybe we want to have the SQL
in a textarea and let it be
modifiable before manually executing?
maybe only when "Auto execute queries" is not enabled?

Config pops out when keybind is pressed,
otherwise it is hidden

Cam we load config options with a file?
So that users that only work with
a specific database don't
need to manually set it each time

```
== UserMessage ==
> List user amount by platform

== Thoughts ==
> The user wants to see the amount of users divided by platform [...]

== SQL ==
SELECT
   platform AS Platform
   COUNT DISTINCT(pseudo_id) as Amount
FROM
   `projectname.dataname_*`
WHERE <date range>

== Result ==
    Platform    Amount
0   Android          5
1    iOS             5
```

Project: projectname

Database: dataname

○ Use Review-agent

☒ Auto execute queries

☒ Save log

if this field is set use this field,
else check env-vars

open log folder?
open the current log?

Command bar with keybinds

Figure 3.2: Initial sketch over the user interface (TUI) that would be developed as one of the implementations.

3.4 Prompt engineering

Prompt engineering was used to shape the LLM to fit the needs outlined for the thesis. This mainly revolved around making sure that it correctly understood the database schema and the characteristics about interacting with that schema.

3.4.1 LLM configuration

For easier debugging and recreating scenarios, the LLMs were configured to be more deterministic and to have fewer variations in their answers. This was done by setting a static seed, a low temperature value of 0, and a Top-P value of 1.

3.4.2 System prompting

The LLM was prompted with a system prompt that included instructions for how to use structured output so that the LLM responded in a JSON format where the pure SQL-query could easily be separated and extracted from the text reply that contained the thought process. This structured output can be seen in appendix C, figure C.2, where the structured reply can be seen in the agent reply and as a filtered message after the summary of just the SQL-reply. The unstructured variant can be seen in appendix C, figure C.1. The model was also instructed to end it's replies with "TERMINATE" once it believed that the SQL-query was complete, signaling to the framework that the chat can now be closed. As the database contained data up to a year old, instructions for how the LLM should limit based on a range of dates in order to keep processing costs down were included in the system prompt.

3.4.3 Role prompting

The prompt instructed the LLM to act as an experienced data analyst with a proficiency with BigQuery and GoogleSQL, which was included as a complement to the system prompt. Further personality traits such as commenting queries to help coworkers understand what each part does, and that the model should not make assumptions about the database schema. The context prompt helped ensure the model thought about creating queries from a data analyst's perspective and make the queries more maintainable and easier to work with.

3.4.4 Context prompting

The contextual prompt, the name for the database, the name of tools available and what the tools do included important imformation required for the LLM to create proper SQL queries.

3.5 Database

The database used, BigQuery, was large and contained several fields. To properly manage the datasets, the tables needed to be manipulated and configured to fit the LLM in use. As BigQuery is hosted online, a connection is needed between the AI agent and the database.

3.5.1 Schema extraction

In order to provide the AI agent with the correct context for the database schema, the schema needs to be generalized and morphed into a format that an LLM can understand. The database also had nested structures, which meant that the format for the LLM also needed to support nested elements. Ultimately, a JSON schema was chosen as the format to use for the LLM.

The source code for mobile applications was sought through to identify what was logged and how it was logged in the database. This was first compiled into a YAML file and comments were added with the relevant information found. The YAML file could later be converted to a JSON schema.

3.5.2 Data fields interpretation

Most data fields have a generic name with a prefix and sometimes a suffix. Without proper domain knowledge of the products, it is difficult to understand what data is being captured and for what purpose. To help the LLM understand what each field represents, a descriptor was included for each field in the JSON-schema that concisely describes what data the field is capturing.

3.5.3 API Connection

Connection, access and privilege control to the BigQuery database was done through the Google Cloud CLI tool [53].

3.6 Framework selection

There were several candidates for a framework to build the prototype with. The candidates were first selected by reading some of their documentation and looking at some examples of projects. After the first round of selections was completed, a quick prototype was built in which two agents collaboratively counted from 10 to 0. The frameworks were then evaluated on the basis of their first impression when used for development.

3.7 Testing

As LLM output is generally non-deterministic, and the database that is queried is continuously updating, it is difficult to structure automated testing. Manual comparison of queries and manual testing were, therefore, the main modes of testing. With access to saved project queries and queries written by experienced engineers, a prompt to the prototype could be reverse-engineered by comparing data results from querying the database. If the resulting data matches; then the prototype passed the test. This form of testing is labor intensive and requires manual work conducted by a human operator to select prepared queries to go through iterations of prompts that match the output, and then to compare and evaluate the results. A second LLM could be introduced, with the capacity to decide if the intent of a generated query matches a manual query that was written in advance.

3.8 Evaluation

The prototype was evaluated through user feedback and user testing. Stake-holders who were interested in using the prototype for their work were tasked with trying it out and later queried for their impressions. User-based feedback is not without flaws, users tend to overestimate how useful AI-powered tools are [54]. User feedback also useful for ensuring that the everyones vision and goals of the prototype aligned. Seven guided questions were provided to the testers to help them write their feedback.

- How was the installation and onboarding process?
- Do you prefer the CLI or the TUI implementation?
- How does it fit your data analysis workflow?
- How do you use the reviewer? Always, never, sometimes, depends.
- How would you describe the quality of the generated queries?
- Do you believe that the agent has an understanding of the database structure?
- Does it meet your expectations for a Natural Language To SQL agentic tool?

In addition, a select number of manually constructed SQL queries were chosen as a baseline to test the agents capabilities against. If a fairly complex, previously written query could be reproduced through natural language prompts, it would seem plausible that an engineer could write a prompt instead of writing SQL.

Automated evaluation was not considered as LLMs can produce different responses from the same initial prompt. The generated SQL query might also use faulty syntax or be logically correct based on the users prompt while still not matching the user's expectation of what should have been generated.

3.9 Literary source review

The technical background is based on the information found in the sources. Whenever information is sourced, the validity of that information must be considered. During the literary study, several sources were read and some had obvious flaws as a scientific source, such as being posted on an anonymous blog or an online forum, while some sources had every mark of being a credible source but lacked basic proper grammar and punctuation, discrediting the source.

3.9.1 Reports

Source [11] is published as a white paper published with Google. It is important to note that it is a white paper based on a product that the publishing company is offering to consumers. The purpose of the paper is to instruct consumers in how to get the best result out of their product, improving the perception of their product. Although white papers are usually related to a product or a company, they tend to be factual and a good source of information.

3.9.2 Books

Sources [22], [50], [51], and [52] are all books. The sources [22], [51], and [52] are all from well-established publishers of academic literature. They have a long history of publishing and are generally seen as very reliable sources. The source [50] is a self-published book, but it is fairly well known in entrepreneurial circles.

3.9.3 Inproceedings

Sources [25] and [37] are inproceedings. They are usually published as part of a conference; they do not need to be peer-reviewed. Both conferences from which the proceedings are published seem like reputable conferences on the basis of their structure, organization, backing sponsors, and accepted talks and papers.

3.9.4 Articles

Sources [13], [14], [26], [27], [39], and [54] are articles of differing sources. Some are from peer-review journals, while others are preprint drafts. The preprint drafts are yet to be peer-reviewed and finalized, but the backing organizations are credible (Nvidia for article [27] and Microsoft for article [39]).

3.9.5 Online sources

Sources [1]-[10], [12], [15]-[21], [23], [24], [28]-[36], [38], [40]-[49], [53], [55], [56], [57], [58], and [59] are all online sources. They are mainly from major and well-known corporations, industry leaders in cloud and AI such as IBM, Google, and Anthropic. They are published as documentation or educational content, creating potential future customers and users of their products.

3.10 Usage of AI

During this thesis, AI tools such as Writefull and Mistral were used to correct and improve spelling, grammar, and sentence structure. AI tools such as Claude and Mistral were used as a code-assistant during programming, helping with refactoring code and proper syntax usage. The authors affirm that the work presented in this thesis was done by the authors.

Chapter 4

Analysis

Several decisions were made, and complications arose during the completion of this thesis. In this chapter, the major decisions, complications that surfaced and how they were managed will be presented.

4.1 Interview summary

The interviewees work across several teams in the Mobile Apps department, with about half also being part in an inter-team QA group that coordinates QA efforts. The roles span the development of VMS applications, testing, and managerial duties. Many would like to make more data-driven decisions, but day-to-day work with data is mostly limited to using pre-existing analytical tools and previously saved queries.

When they interact with data, they typically rely on analytical views created in LookerStudio and saved queries in BigQuery, often creating new queries by mixing and modifying existing ones. The general impression seemed to be that it is uncommon to create brand new queries and analytical views. When and if the need arises, it can take anywhere from 15 minutes to a full workday depending on the complexity of the query and the persons experience. The retrieved data are usually kept in their raw format (as tables or text) and rarely morphed into visual forms, such as graphs. The data is used to gain a better understanding of common usage patterns and guide feature decisions related to app functionality.

There are several recurring barriers that make new data analysis work difficult. Some primary issues are data collection and schema complexity; certain data that would be useful for insight might not currently be gathered or is logged in a way that is inconvenient to use. The schema of the database is one large table with several nested fields, which increases the mental overhead needed when working with the table and creating queries. BigQuery also uses a dialect of SQL called GooglesQL which, while mostly compatible with traditional SQL, has some unique features. As queries are not creating frequently, the knowledge and experience erodes, meaning that time must be spent to re-familiarize oneself with the SQL dialect and the table structure.

Interviewees often check if a query, or a similar one, has been created before, otherwise they might try to ask a colleague (often student workers that work part time with mostly data assignments) for help. If no previous query has been created, or the colleague is not available, the goal of creating a new query is often abandoned, as the time and effort needed to create it is usually not worth the data retrieved.

The interviewees who have worked with BigQuery describe it as powerful but unintuitive. The SQL dialect adds functionality but also leads to confusion. There is a lack of visibility into the fields that exist and what they represent. More up-to-date documentation of the table schema, with explanations of what data are stored and why it is stored would speed up query creation or enable broader and more exploratory queries.

Finally, the interviewees answered some questions about AI tools. Adoption varies between interviewees. Organizational rules dictate how AI tools can be used and which tools are sanctioned for broad usage. Common use cases include information and documentation lookup, "rubber-ducking" debugging (explaining the code and / or bug to a "rubber duck", in this case an LLM, and collaboratively work to find and solve the bug), and code-assistance. The use of fully autonomous agents is rare, the interviewees mostly shared a concensus that the technology being new and not fully mature or capable yet.

4.2 Scope and identified needs

After the interviews were conducted, the main scope for the prototype and the needs of the stake-holders could be decided. The main focus of the prototype would be to create SQL queries from NLP. This was identified from the interviews as a feature that would be utilized and could bridge the gap between the current data analysis work being done and the data analysis work that they want to do.

There was no significant desire for the prototype to be able to create visualizations of the data retrieved. Therefore, that feature of the prototype would not be developed; instead, time was allocated to improve the user experience. A simple visualization in the form of a table was still included in the final prototype. The revised GANTT chart can be seen in figure 4.1. In addition to pivoting from data visualization, as the presentation date was set, a week of report writing was changed to presentation preparation, and a week of testing was removed to better match the set dates.

4.3 Framework

The AutoGen framework [35], was tested as a suggestion of one of the company supervisors. It is backed by Microsoft and was very quick to get up and running while still being extensible. It was a great framework for quick prototyping and future-proofing.

4.3.1 AutoGen-split

In late 2024, the original AutoGen developers forked the AutoGen repository at version 0.2 and started developing AG2 (AutoGen 2). Microsoft rewrote their codebase and released AutoGen 0.4, which had a new syntax and structure. The

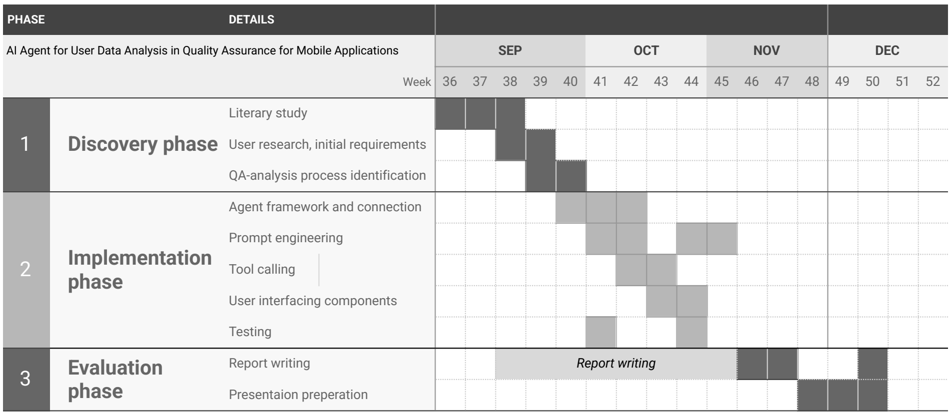


Figure 4.1: Revised GANTT chart after interviews were held.

AG2-team kept the Python package named "AutoGen" while Microsoft released their rewrite as "AutoGen-core" and "AutoGen-AgentChat".

This split has resulted in difficulty finding the right documentation and help online, as any resources from before 2025 are using the old syntax. The decision was made to continue using the new Microsoft version instead of the AG2 version. Although AG2 had more resources available online, the Microsoft documentation appeared more refined.

4.3.2 Semantic Kernel

Semantic Kernel is another AI agent framework by Microsoft, offering production-ready AI agent tooling [55]. As part of AutoGen's rewrite, it was rewritten to better align with Semantic Kernel and with the goal of being able to integrate AutoGen agents into the more mature Semantic Kernel [56].

4.3.3 Microsoft Agent Framework

In late April 2025, Microsoft also started releasing their Microsoft Agent Framework *(MAF)*. It is currently in public preview but is the designated sucessor to both AutoGen and Semantic Kernel [57]. There are already migration plans written from AutoGen or Semantic Kernel. Once MAF is mature enough, it will be possible to migrate to it and continue development on the prototype [58].

4.4 Large language model

There was a strict requirement to use a local LLM for the prototype. This meant that frontier models were not available for use. The models that could be used were older, had less training data, and struggled with some cognitive tasks that the frontier models were more likely to succeed in. Providing more structure with system prompts, tool calls, and a well defined schema over the database structure was needed to help the local model perform to the best of its abilities. A local model that was trained or fine-tuned on the specific data would likely perform better, but that was outside of the scope of this thesis.

4.5 Database

The database is a large BigQuery table; there are a lot of nested values and null entries that make it difficult to view the entire table at once. To help the agent navigate the table, a schema representation was constructed.

The addition of pipe syntax to BigQuery was made in late 2024 [59]. Although the pipe syntax could simplify queries and help reduce logical mistakes made by the agent, the late addition meant that it was not included in the training data for the model used. Instead, the traditional form of GoogleSQL had to be used, with the model creating common table expressions and subqueries.

4.5.1 Schema design

The schema was designed in a JSON format. This format supports nested objects and can properly represent the database schema, but JSON can be difficult to read with large nested objects. The agent will parse it to a Python dictionary with the tools, but the structure makes it difficult to modify and expand on the schema. Some values in the database are straightforward, an integer value for a duration column, while others are a set of possible string values; used like enums. This meant that the schema also needed to include possible values for fields that had enum-like definition. The JSON schema was eventually constructed as a two-object JSON file. The first object contained all columns in the database. It represented the structure of the table, what fields existed, and where they were located. The second object held all unique events that were logged and their specific key and value parameters. These parameters would vary between all events, so they needed to be manually checked and added. The structure of an event, with possible values for the key "reason" can seen in figure 4.2.

4.5.2 Source code logging

The source code for both the Android and iOS versions of the app was analyzed to find definitions of logged events so that a description could be added to help the agent navigate the schema. The logging approach varied slightly between the iOS and Android versions and there were also variations between different source files for the same platform. Different developers have had their own interpretation of how and what should be logged. There were some parts where developers had logged under a key "some_name" instead of the more common "somename", the added underscore led to the same log event being logged in different places in the database. Solving this problem was outside the scope of this thesis, and instead the JSON schema had to be planned around this; the Android app was newer, so it was decided that it would be used as the source of truth for events that were shared between iOS and Android. If uncertainty ever arose, a colleague would be consulted to point out which description should be used.

```
{
    "name": "audio_transmission",
    "description": "Log audio transmit with duration and possible error reason.",
    "params": [
        {
            "key": "status",
            "type": "string",
            "values": ["success", " error"],
            "description": "The status of the operation."
        },
        {
            "key": "reason",
            "type": "string",
            "values": [
                "transmit_client_failure_20005",
                "transmit_client_failure_20000",
                "transmit_client_failure_10002",
                "transmit_client_failure_10000",
                "transmit_client_failure_20004"
            ],
            "description": "The reason for the error."
        },
        {
            "key": "duration",
            "type": "int",
            "description": "The duration of the audio transmission."
        }
    ]
},
```

Figure 4.2: An event entry in the agent's JSON schema. An entry contains parameters that can be accessed when unnesting the table on that specific event. Fields also contain a type, describing the datatype that is stored, and a description that informs the agent of what is stored so it can draw conclusions for when to use a certain event.

4.6 Implementation

The work of developing the prototype was divided into logical steps, making sure that the connection to the LLM worked, letting the LLM respond to user input, and then working with the agent framework.

4.6.1 Defining a model client

AutoGen-Agentchat expects a model client. This sets the structure of the LLM model that is used. If the model used is capable of using OpenAI-like API endpoints, an OpenAIChatCompletionClient can be used. In the model client, the model name, the API url, the API key, and configuration parameters such as temperature and Top-P can be set. Using a model with OpenAI-like API endpoints allows for more flexibility with the model selection. It is possible to switch out different models to power the agents as long as they have an OpenAI-compatible API. If a custom model is used, the specific capabilities of that model must be defined in a dictionary called model info, since the framework expects an OpenAI model. The model client declaration can be seen in appendix D, figure D.1. This flexibility in model selection was a clear reason to use a model that was compatible with the OpenAI API.

4.6.2 Configuring an agent

Once the model client is defined, an agent can be configured. In this work, AutoGen-Agentchats AssistantAgent was used. This is an agent with basic agentic capabilities such as the use of tools. For more extensibility and customisability, AutoGen-Core can be used instead. With AutoGen-Core, a fully custom agent can be defined, allowing endless possibilities. Consequently, the drawback of using a Core-based agent is that it is initially extremely limited. In order to read messages and output messages, functions need to be defined and written. The AssistantAgent was deemed suitable for this thesis, as it already had the functionality to interact and generate new messages, together with an easy way to write and connect tools. The configuration of an AssistantAgent can be seen in figure D.2.

An agent can also be defined to use a structured output if the underlying LLM supports structured outputs. With a structured output, it is easier to get a generated response in a specific format. For the SQL-writer agent a structured output was used, where the agent divided its response into a JSON message with one part as the text reply and the other pure SQL. The code for this can be seen in figure D.3.

The agent responds with a TaskResult-class. The TaskResult has several properties that can be accessed, such as a list of all messages sent. The code for extracting SQL from a structured output called AgentResponse can be seen in figure D.4. The last message in the list of messages is accessed, and the property of the SQL response in the content is printed on the console. Returning the TaskResult and not only the SQL reply means that all messages can be logged, not just the final generated SQL query. This functionality is important when it comes to logging.

4.6.3 Tools

Tools are implemented as functions, which subsequently are wrapped in a AutoGen-Core FunctionTool-class where a description is added. This description tells the agent what the tool is used for and what it does. In figure D.5, the tool that the agents use to view the database schema can be seen. The tools are then listed in the tools parameter when declaring an agent, as seen in figure D.2. This approach made it easy to share tools between agents and add or remove tools without much need for modification of the agents themselves.

The tools are not dependent on the type of agent framework used; rather the single requirement is for the LLM to be able to separate its generated SQL from other generated text. As long as the LLM used supports a form of strucutred output, the tools can be used. For this reason, a choice was made to develop small tools that fit the prototypes needs rather than using MCP based-tools. While MCP based-tools would have been more extensive, the added complexity would increase development entropy.

4.6.4 Single and multi-agent teams

There are pros and cons to using a single agent compared to using a team of several agents. A single agent can produce results faster, but is limited in its level of scrutiny if it is asked to review its own work. Adding another agent to act as a reviewer means that the reviewer does not necessarily have the same context window as the original agent. Thus, the reviewer can observe the generated message without any pre-existing context. This does increase the run time as there now are two models that need to be instanciated and fill their context before producing output. This comparison can be seen in table 4.1, where a single and a two-agent team were both prompted with the same task.

AutoGen supports several different forms of group chats that can be used. RoundRobinGroupChat lets all participating agents and users take a turn sending a message to the rest of the participants. As the prototype revolved around SQL generation, reviewing and correcting in a loop until the reviewer accepts the generated SQL query, a RoundRobinGroupChat was utilized for its ease of implementation and speed. More complex group chats such as SelectorGroupChat and MagenticOneGroupChat, where an LLM makes decisions on which participant should get a turn, were considered but ultimately deemed superfluous.

The definition of the team used for this prototype and the initialization of the team can be seen in figure 4.3. The condition for ending the conversation was set to a maximum of 20 turns, where each participant had sent 20 messages, or when one of the agents responded with *"TERMINATE"*, as the SQL-writer agent is instructed to do in its system prompt.

Table 4.1: A comparison of the execution time and the amount of generated messages from a single and a two-agent team configuration receiving the same prompt.

Nbr of agents	**Nbr of messages**	**Execution time** (s)
1	4	8,31
2	8	18,1

```python
from autogen_agentchat.teams import RoundRobinGroupChat
from autogen_agentchat.conditions import
    TextMentionTermination
from autogen_agentchat.messages import StructuredMessage
from autogen_agentchat.ui import Console


team = RoundRobinGroupChat(
        participants=[sql_writer, sql_reviewer],
        max_turns=20,
        termination_condition=TextMentionTermination("
            TERMINATE"),
        custom_message_types=[StructuredMessage[AgentResponse
            ], StructuredMessage[ReviewerResponse]],
)

result = await Console(
    team.run_stream(task=prompt),
)
```

Figure 4.3: An AutoGen RoundRobinGroupChat being defined with participating agents, the maximum amount of turns each agent can send a message, custom outputs for agents, and a condition for ending the team. The team is then initiated with a task.

4.6.5 Streaming

A console can be used when starting a task for the agent(s), acting as an output for the messages that are being sent. By enabling streaming when configuring the AssistantAgent and when starting a task in the console, each message produced by the LLMs is streamed out in the console as they are being generated. This lets users see the process of message generation and see when LLMs get stuck in their message generation. It is also easier to see if there are availability or network issues with the LLM connection when using streaming, as the chunks of generated messages will be larger per chunk and also take longer to appear.

Integrating streaming with other parts of the prototype, such as the TUI, proved difficult. A decision was made to scrap the streaming functionality and instead focus on more integral parts. In place of streaming, the TUI presents a loading indicator to inform the user that a task is currently being processed.

4.6.6 Logging

Each time the agent is run, AutoGen logs the entire process, including technical parts such as networking information. These logs are useful for debugging and give insight into every detail, but they are extremely dense and difficult to read. For that reason, a custom logging solution was developed in which the initial prompt, the generated messages, and certain values such as configuration settings and the time it took to generate a query were saved. These output logs act more as a historical view of the prompts made and the resulting output, which users can look back on. Once the prototype became stable enough, the built-in detailed logging was disabled, and instead the custom, streamlined, logging solution was utilized. Together with custom scripts, these logs could be quickly parsed for analytical data on usage and performance. An example output log from the custom solution can be seen in appendix E, figure E.1

4.6.7 BigQuery connection

Authentication to BigQuery was done using the Google Cloud CLI tool. This tool uses the users own credentials to authenticate them and limits connection to projects that the user is authorized to use. This approach is safer than using an application-based API key as there is no need to expose an API key and risk unauthorized intrusion through that key. Instead, it is up to each user and their team to manage the authorization.

Through the Python module for Google Cloud, a BigQuery library could be imported. This library was used to choose the project and the data set to send queries to and was responsible for sending queries to BigQuery and retrieving the results. Once retrieved, the results could be parsed to a Pandas dataframe, which enabled more functionality, such as saving as a CSV-file or transforming the resulting table. The BigQuery library sends raw SQL prompts. This does pose a risk for SQL injection, but the interface for loading SQL queries is not available to the user, instead it is the agent that loads and executes SQL queries.

To avoid running a very large and expensive query, a dry-run is first attempted which returns the amount of bytes that would be processed, together with a syntax validation check. If the processed amount is under the allowance and the syntax validation passes, the query can be sent to BigQuery.

4.6.8 Argument parsing

Sometime there is only a need to generate a query, not to get the result right away. For that reason, the user can add arguments when starting the agent, for example; adding the "-e" flag will enable query execution, allowing the agent to automatically execute SQL queries to BigQuery. Adding the "–ui’ flag will start the TUI-interface. There are more argument flags for the prototype; this approach is flexible and lets the users pick how they want to use the agent without adding much development overhead. The arguments available to the users can be seen in figure 4.4.

```
$: btma --help
    usage: btma [-h] [--ui] [--reviewer] [-e] [-p PROMPT] [-c]
         [-l] [--delete_dirs]

    BTMA - Bachelor's Thesis Mobile Apps

    options:
      -h, --help            show this help message and exit
      --ui                  Start with UI
      --reviewer, -r        Enables reviewer agent.
      -e, --autoexec        Automatically executes queries and
          return result
      -p PROMPT, --prompt PROMPT
                            Prompt to send to the agent
      -c, --copy            Copies the generated SQL query to
         clipboard
      -l, --list_dirs       List the directories and folders
         that are created by the app
      --delete_dirs         Deletes directories created by the
          app
```

Figure 4.4: Possible argument flags that can be used when starting the prototype. Settings such as enabling a second agent to utilize a team structure, automatically executing the generated SQL to BigQuery, and starting a graphical user interface can be called.

Chapter 5

Result

This chapter contains the results of the thesis conducted. The protoype's capabilities are outlined, and measurements that were taken are presented.

5.1 Prototype

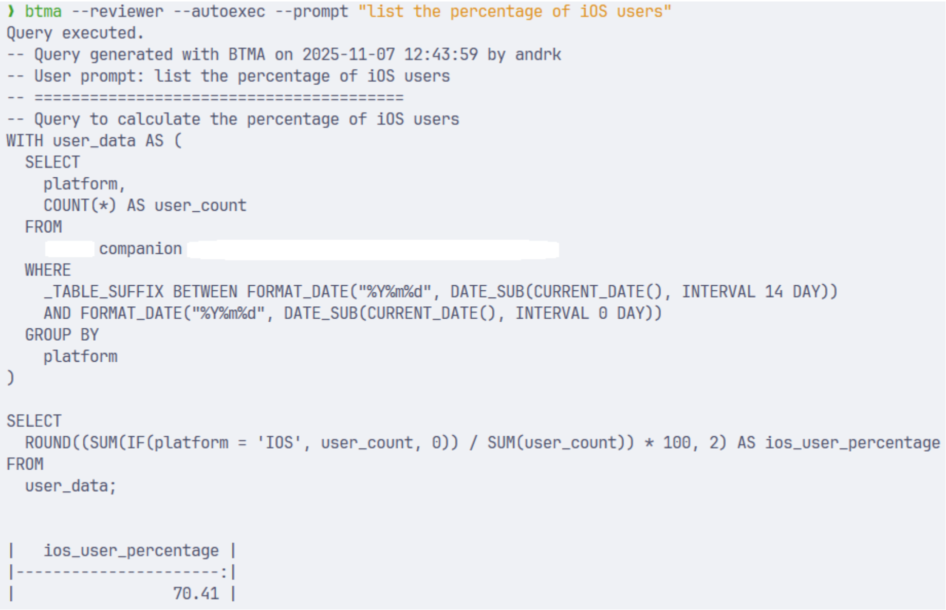
An AI agent prototype was completed. This prototype generates SQL tailored to a specific project database from a users natural language prompt. With added contextual knowledge of the database's structure and descriptive documentation about what is stored in the database, the prototype can draw conclusions from prompts and generate queries that require domain knowledge of the project, with differing levels of correctness.

The prototype supports multiple forms of interaction and implementation into end users' personal workflows. With exception to the initial installation and setup, a user that prefers a graphical interface can rely entirely on the TUI. While a user that has created a personally custom terminal-based workflow can implement the CLI implementation with argument parsing and embed the prototype into their own terminal scripts and other programs. Screen captures of both the CLI and the TUI implementations can be viewed in figure 5.1 and 5.2, respectively.

5.1.1 Tooling

The prototype supports loading a database schema representation in JSON. This is the only tool needed for the agent to understand the structure of the database and the contextual domain knowledge required to form project specific queries from initial prompts. A workflow has been created to quickly prepare schema representations of other databases, allowing for future extensibility of the prototype.

If the user wants to quickly see the data that the generated query would retrieve, they can execute the SQL with the prototype and get a table representation of the data retrieved, as well as a locally saved csv-file for further use. This tool also has built in guardrails to mitigate run-off costs with large or expensive queries being executed in BigQuery. The user can set the limits of the processed data size and cost estimation in their configuration file.

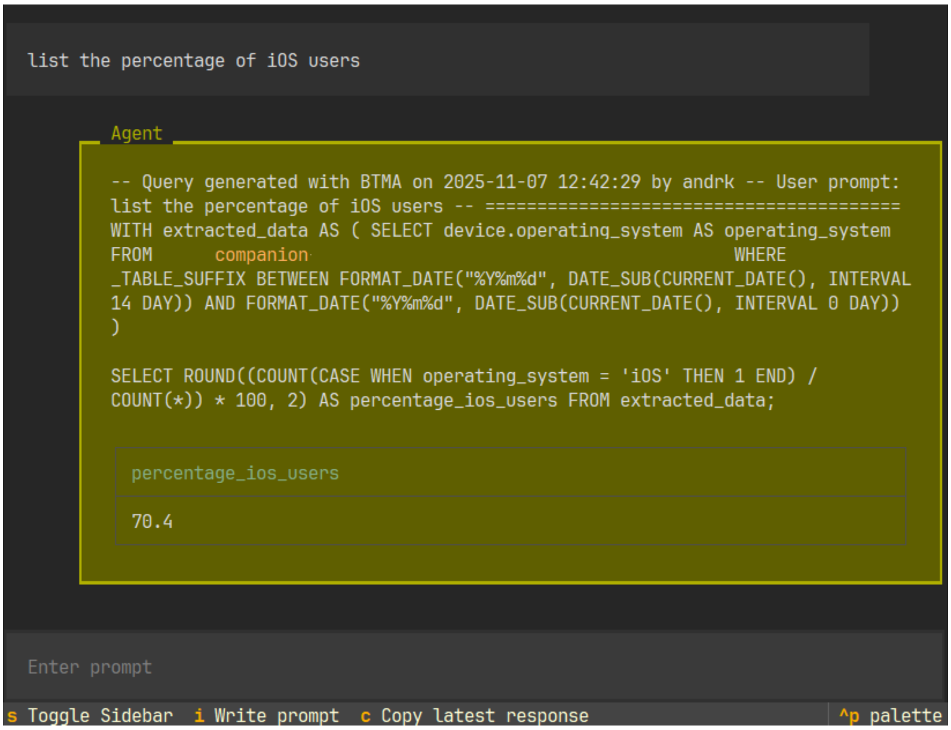


```
❯ btma --reviewer --autoexec --prompt "list the percentage of iOS users"
Query executed.
-- Query generated with BTMA on 2025-11-07 12:43:59 by andrk
-- User prompt: list the percentage of iOS users
-- ========================================
-- Query to calculate the percentage of iOS users
WITH user_data AS (
  SELECT
    platform,
    COUNT(*) AS user_count
  FROM
    [redacted] companion [redacted]
  WHERE
    _TABLE_SUFFIX BETWEEN FORMAT_DATE("%Y%m%d", DATE_SUB(CURRENT_DATE(), INTERVAL 14 DAY))
    AND FORMAT_DATE("%Y%m%d", DATE_SUB(CURRENT_DATE(), INTERVAL 0 DAY))
  GROUP BY
    platform
)

SELECT
  ROUND((SUM(IF(platform = 'IOS', user_count, 0)) / SUM(user_count)) * 100, 2) AS ios_user_percentage
FROM
  user_data;


|   ios_user_percentage |
|----------------------:|
|                 70.41 |
```

Figure 5.1: A screen capture of the CLI implementation of the prototype being prompted with the argument flags for adding a reviewer agent and automatic execution of the generated SQL to BigQuery enabled. The generated SQL is printed to the terminal and the retrieved data is presented in the form of a Markdown table after the SQL.



```
list the percentage of iOS users
```

```
-- Query generated with BTMA on 2025-11-07 12:42:29 by andrk -- User prompt:
list the percentage of iOS users -- ========================================
WITH extracted_data AS ( SELECT device.operating_system AS operating_system
FROM      companion                                         WHERE
_TABLE_SUFFIX BETWEEN FORMAT_DATE("%Y%m%d", DATE_SUB(CURRENT_DATE(), INTERVAL
14 DAY)) AND FORMAT_DATE("%Y%m%d", DATE_SUB(CURRENT_DATE(), INTERVAL 0 DAY))
)

SELECT ROUND((COUNT(CASE WHEN operating_system = 'iOS' THEN 1 END) /
COUNT(*)) * 100, 2) AS percentage_ios_users FROM extracted_data;
```

percentage_ios_users
70.4

Figure 5.2: A screen capture showing the graphical TUI-implementation of the prototype being prompted with a second agent and automatic execution of generated SQL to BigQuery enabled. The generated SQL is printed to the screen in a chat-style interface and the retrieved data is presented in a rendered Markdown table after the SQL.

5.2 Reverse engineering queries

Eight project queries, queries that had been saved to the database by other developers, were chosen as a target. The goal was to recreate the result of the retrieved data that the project query retrieved in a maximum of five prompts to the agent. The prompts started out as abstract and containing little information. Details and information were then added onto the prompt, evolving it and adding complexity, simulating an experienced developer with extensive domain knowledge writing a prompt. For some prompts, certain field values and descriptions did not exist or were incorrectly described in the database schema representation, which consequently meant that descriptions of these values and fields were included in the prompt to guide the agent.

In total, six of the eight queries were considered successful. The retrieved data might not be 100% consistent with the project query, but were similar enough to consider them correct. The amount of prompts needed for each project query and a short description can be seen in table 5.1. The two failing queries may have several underlying causes. With only the raw SQL available, the intent, motivations, and goals of the original author had to be inferred in order to construct an accurate natural-language prompt. Some project queries were several years old, and logging practices had changed over time. Because the AI-assisted schema extraction considered only data logged in the previous 24 hours, any fields or events not logged during that period were excluded from the generated schema. The failures can therefore be attributed both to limited insight into the original querys intent and to an incomplete database schema.

Table 5.1: Reverse engineering results with a query identifier, the amount of prompts sent for a specific query, and a short review of the resulting SQL and retrived data.

Query	Prompts	Successful
1	3	Yes, a null-case wasn't handled
2	5	No, could not complete the calculation
3	3	Yes, 2nd prompt hallucinated
4	3	Yes, had to provide column names in the promt
5	4	Yes, difference in how values were rounded
6	2	Yes, almost correct with first prompt
7	5	No, wrong values and syntax errors
8	1	Yes, correct on first prompt

5.3 Efficacy

As seen in table 5.1 and in section 5.2, the efficacy of the prototype is passable. It struggles with complex queries, and the database schema representation also contains flaws due to factors such as a lack of knowledge of what fields means, and data being logged incorrectly from the source code. The syntax errors that do show up in the generated queries tend to be quick to manually correct, usually missing an unnest or misinterpreting a value type. Even when the prototype gets the query wrong, it still provides a large chunk of the work correctly done, meaning that an engineer can focus on fixing the remaining parts of the query. For simpler queries, such as the ones in fig. 5.1 and fig. 5.2, the prototype is correct with exceptions far between, and those exceptions attributed to incorrect schema representation.

5.4 Efficiency

The prototype's speed is largely affected by the underlying model and the hardware on which the model is hosted. However, regardless of model and hardware selection, a trend can be seen that shows how much time is added to the execution time when a second agent is added to review the generated queries.

A total of 206 logs were analyzed, 103 of those running with a review agent enabled and 103 with a single agent. The average time per prompt for a single agent-team and a two agent-team can be seen in table 5.2. Adding a reviewer to the team increases the average execution time by 260%. This increase is inflated due to the ability of a reviewer to deny the initial generated query, forcing the SQL-writer agent to create a new query. In table 5.4, there is a 72 second addition when the reviewer denies the first iteration. This means that by adding a reviewer only, the execution time increases from 15.1 seconds to 38.1 seconds, a 150% increase in execution time. The worst case scenario that has been logged would be when the reviewer denies the query, resulting in a 72 second and a 480% increase in execution time compared to when using a single agent, without a reviewer.

In table 5.3, the appearance of different amounts of messages that are generated can be viewed. The amount is always odd, since the writer-agent always has the final message. For a single agent-team, it sends a single message, while for a two agent-team, the writer sends the first message and waits for the reviewer to respond before the writer sends a final message if the query was approved, or completing another round where each agent sends another message. A second iteration of the generated query was required 33% of the time when prompting with a reviewer enabled. This means that most of the time, only a single iteration is required and there is not a large increase in execution time to enable the reviewer agent.

All of these values that have been measured depend on several factors. The underlying model host might have been under load for certain prompts, which would increase the execution time. The user might only prompt for easy queries, or the prompt includes the information needed for the agents so that they do not need to draw their own logical connections between different columns in the schema representation. These results can only be used as an indication that adding a second agent increases the execution time and roughly the size of the increase one can expect.

Table 5.2: Amount of logs analyzed for different team constellations and the average time per prompt in seconds.

Team structure	**Amount of logs**	**Average time** (s)
Single agent	103	15.1
Two agents (reviewer)	103	54.5

Table 5.3: Occurance of amount of messages sent between agents for a prompt.

Messages sent	**Count**
1	103
3	69
5	34

Table 5.4: Average time per prompt for different amount of messages sent between agents

Messages sent	**Average time** (s)
1	15.1
3	38.1
5	87.9

5.5 Prompting

After reverse-engineering some queries, it became prevalent that prompting had a significant impact on the result. Adding context and some SQL specific details, such as including specific names of which fields are supposed to be queried or specific SQL functions that should be used, helps the agent properly create a query that matches user expectations.

5.6 Feedback

Some experienced engineers who occasionally write queries were asked to test the prototype and provide some feedback. They believed that the installation and onboarding process was easy to follow with the accompanying documentation. They felt aligned to use the TUI implementation over the CLI, this was attributed to factors such as the TUI having a familiar interface and a prominent loading indicator while waiting on the generated query from the agent. Some details tend to be missed in the generated query, such as missing a specific event or misinterpreting what event should be queried, but it is usually a quick fix for experienced engineers. The query generation is fast, and being able to focus on the data retrieved instead of writing SQL syntax fits nicely into their work. All testers use the reviewer agent; since the work is completed in the background, the added execution time does not bother them, and they value the prospect of a correct query more. The prototype understands the database well enough to be of use, some mistakes are made, but not in a prohibitive manner. Overall, the prototype meets their expectations for a tool to help with generating queries based on a project database.

Chapter 6

Conclusion

This chapter consists of a concise review of the thesis, answers to the research problems, future developments, and thoughts from an ethical perspective.

The prototype facilities a line of communication between engineers and an AI agent, allowing for SQL generation with efficacy and efficiency. The prototype understands the project database as an entity and can mostly make decisions regarding which fields are needed in order to create a query in a time-efficient manner. With features for configurability for users, such as choosing between a CLI or TUI interface, enabling a reviewer agent, and the automatic execution of queries to BigQuery, users can fit the prototype into their own personal workflows and use the prototype in a manner that fits their needs.

6.1 Research problems

The prototype was developed with the research problems (1.4) in mind. Here, a reflection will take place, asking if the research problems were answered.

1. **How does Axis currently analyze user data for its mobile applications?**

   Answered through interviews. User data is mostly analyzed through prepared queries and presented in LookerStudio. It is uncommon to allocate time to write new queries for quick data lookups.

2. **What are current constraints limiting user data analysis?**

   Answered through interviews. Some queries can take hours to write, and it is difficult for engineers to find time in their day to work on queries. There is also a requirement of extensive domain knowledge of what is being gathered and how it is being stored in the database, currently there is no complete documentation covering this. The GoogleSQL dialect of SQL adds some features and behaviors that are unknown to most developers, even with SQL experience.

3. **Where can an AI agent be implemented within a user data analysis-centric workflow?**

As writing queries arose as a hindrance to user data analysis, it quickly became the target of this thesis. An AI agent could be implemented in a fashion to help engineers write their queries. This can be achieved either in a question-and-answer based fashion, where the engineer is responsible for writing most of the SQL but can ask the agent for what fields to use depending on what data is sought after, or it can be implemented in a configuration where the agent is responsible for writing the queries, allowing engineers to focus solely on describing what they are searching after. The latter description was the targeted implementation for this thesis.

4. **How can a prototype AI agent be implemented?**

   A prototype can be implemented using an agent framework to provide scaffolding when developing. The prototype could then be fashioned into single agent or multi agent-teams, where agents collaboratively work together to create a query from the initial prompt by the user. With a focus on enabling users to fit the prototype into their personal workflows, the prototype is more adoptable as a concept for users. Through argument parsing for the CLI implementation, an experienced user can use the prototype with other tools and scripts while more novice users, or users who prefer a visual interface, can interact with the TUI implementation for a recognizable chat interface.

5. **How can an AI agent visualize data and present results?**

   This research problem was largely left unexplored after the interviews were conducted. It was not deemed as functionality that would be utilized. Although not explored, the prototype provides the retrieved data in table format and as a file with comma-separated values. The table format is easy to view and distinguish between different values, while the comma separated values file is suited for use with other tools, such as a spreadsheet viewer.

6. **How can a prototype AI agent be evaluated?**

   Evaluation of an AI based system is difficult. AI tends to respond with slight differences, even to the same input. The agent also interacts with a live database that is updated during the day, so it is not possible to compare the retrieved values to see if it matches the expected value. Instead, a form of evaluation was performed in which queries written by engineers were reverse engineered by writing a prompt that retrieved the same or similar data. This evaluation was not without faults as it depended on the authors prompting techniques and domain knowledge of what data should be retrieved. However, it served as a measure to see if a prompt could achieve the results that engineers have previously queried, and the results were good. The prototype was mostly capable of writing a query that got most, if not all, of the data correct in less than 5 prompt attempts.

6.2 Future developments

The current iteration of the prototype only supports a single project in BigQuery. This also means that only one application out of all the applications that are developed is supported. One future development would be to expand the database schema so that other projects are also included and can be selected. This change would enable more engineers on the team to be able to use the prototype in their work.

The prototype is currently limited to a single language model. Testing and adding more models could improve the experience. Smaller models might run faster and still be useful for simpler queries, while larger and slower models can be used for complex queries.

The sole focus of the prototype is to generate SQL, it does not reason about the retrieved result. A large addition would be to let the agent reason about the data, let it act as a data engineer and draw conclusions. Although it might not always be correct, it might expose patterns and notice details that an engineer would overlook. When approaching a problem from an analytical point of view, it is generally a good idea to obtain additional input from more observers.

6.3 Ethical viewpoints

Automation and the adoption of AI technologies can have the consequences of rendering positions obsolete. That means that a number of people can lose their jobs and income, while business owners can reap the rewards of a cheaper workforce that is capable of working around the clock. This can lead to larger gaps in economic inequality that can be seen around the world. The authors of this thesis note that this prototype is not usable without an operator with extensive domain knowledge. As Axis does not currently analyze all data, largely attributed to a lack of resources in manhours and the sheer amount of data, this prototype decreases the time needed for engineers to work with data analysis, letting them do more analytical work and also work with their usual tasks.

Letting an AI work with user data can be seen as a privacy risk. The prototype is not capable of reaching any personal data or data that can be used to identify any specific user. The data available to the prototype are statistical analytical data, device information such as brand and model, anonymized usage patterns, and how users interact with the application.

As the prototype is narrow in scope, it seems plausible that an SLM, below 30bn parameters in size, would perform well. A small model that is trained on the project database and SQL would lower the energy consumption both during training and during deployment compared to large state of the art models, such as Claude Sonnet, ChatGPT-5, and Gemini 2.5 Pro. Letting the user choose when to use a second agent in the team is also a way to reduce energy consumption. The user can be conscious and manually enable the reviewer for larger or more complex prompts and queries.

Bibliography

[1] Axis Communications AB. “We Are Axis,” Accessed: Sep. 17, 2025. [Online]. Available: `https://www.axis.com/about-axis`.

[2] Axis Communications AB. “Why a learning culture is the key to success,” Accessed: Sep. 17, 2025. [Online]. Available: `https://lifeat.axis.com/learning-culture/`.

[3] Axis Communications AB. “Moments that made us,” Accessed: Sep. 17, 2025. [Online]. Available: `https://www.axis.com/about-axis/history`.

[4] Axis Communications AB. “Privacy policy,” Accessed: Sep. 18, 2025. [Online]. Available: `https://www.axis.com/privacy/privacy-policy`.

[5] Axis Communications AB. “What anonymous usage data is collected in Axis Camera Station?” Accessed: Sep. 18, 2025. [Online]. Available: `https://faq.axis.com/s/article/what-anonymous-usage-data-is-collected-in-vms?language=en_US`.

[6] Apple Inc. “AXIS Camera Station Edge,” Accessed: Sep. 18, 2025. [Online]. Available: `https://apps.apple.com/se/app/axis-camera-station-edge/id1385994562`.

[7] Axis Communications AB. “Customer stories,” Accessed: Sep. 18, 2025. [Online]. Available: `https://www.axis.com/customer-story`.

[8] Axis Communications AB. “Global tech leader secures 61 sites on 3 continents,” Accessed: Sep. 18, 2025. [Online]. Available: `https://www.axis.com/customer-story/red-hat-axis-device-manager-global-security`.

[9] Axis Communications AB. “Modernizing Retail Technology Operations to Improve Business Efficiency,” Accessed: Sep. 18, 2025. [Online]. Available: `https://www.axis.com/customer-story/lowes-business-efficiency`.

[10] OpenAI. “A practical guide to building agents,” Accessed: Sep. 11, 2025. [Online]. Available: `https://openai.com/sv-SE/business/guides-and-resources/`.

[11] L. Boonstra, “Promt engineering,” White paper, 2025. Accessed: Sep. 11, 2025. [Online]. Available: `https://archive.org/details/22365-3-prompt-engineering-v-7`.

[12] K. Ichhpurani. “Shaping the future together with our partners: The potential of agentic AI,” Accessed: Sep. 18, 2025. [Online]. Available: `https://cloud.google.com/blog/topics/partners/sharing-new-report-on-the-potential-of-agentic-ai`.

[13] A. K. Pati, “Agentic ai: A comprehensive survey of technologies, applications, and societal implications,” IEEE Access, vol. 13, pp. 151 824–151 837, 2025. DOI: `10.1109/ACCESS.2025.3585609`. Accessed: Sep. 25, 2025.

[14] A. Nordgren, “Artificial intelligence and climate change: Ethical issues,” Journal of Information, Communication and Ethics in Society, vol. 21, no. 1, pp. 1–15, Feb. 2022, ISSN: 1477-996X. DOI: `10.1108/JICES-11-2021-0106`. Accessed: Sep. 22, 2025. [Online]. Available: `https://doi.org/10.1108/JICES-11-2021-0106`.

[15] Engineers of Sweden. “The Ten Principes of the Code of Honour,” Accessed: Sep. 19, 2025. [Online]. Available: `https://www.sverigesingenjorer.se/om-forbundet/organisation/hederskodex/`.

[16] NASA. “What is Artificial Intelligence?” Accessed: Sep. 22, 2025. [Online]. Available: `https://www.nasa.gov/what-is-artificial-intelligence/`.

[17] C. Stryker and E. Kavlakoglu. “What is artificial intelligence (AI)?” Accessed: Sep. 22, 2025. [Online]. Available: `https://www.ibm.com/think/topics/artificial-intelligence`.

[18] IBM. “What is unsupervised learning?” Accessed: Oct. 2, 2025. [Online]. Available: `https://www.ibm.com/think/topics/unsupervised-learning`.

[19] Deeplearning.AI. “A Complete Guide to Natural Language Procedssing,” Accessed: Sep. 28, 2025. [Online]. Available: `https://www.deeplearning.ai/resources/natural-language-processing/`.

[20] C. Stryker and J. Holdsworth. “What is NLP (natural language processing)?” Accessed: Sep. 28, 2025. [Online]. Available: `https://www.ibm.com/think/topics/natural-language-processing`.

[21] Cloudflare. “What is a large language models (LLM)?” Accessed: Sep. 24, 2025. [Online]. Available: `https://www.cloudflare.com/learning/ai/what-is-large-language-model/`.

[22] K. Huang, Agentic AI : Theories and Practices. Springer, 2025, ISBN: 9783031900259.

[23] Amazon Web Services, Inc. “What are AI agents?” Accessed: Sep. 25, 2025. [Online]. Available: `https://aws.amazon.com/what-is/ai-agents`.

[24] Google. “What is an AI agent?” Accessed: Sep. 25, 2025. [Online]. Available: `https://cloud.google.com/discover/what-are-ai-agents?hl=en`.

[25] S. Yao et al., “React: Synergizing reasoning and acting in language models,” in International Conference on Learning Representations (ICLR), 2023.

[26] R. Sapkota, K. I. Roumeliotis, and M. Karkee, “AI Agents vs. Agentic AI: A Conceptual taxonomy, applications and challenges,” Information Fusion, vol. 126, p. 103 599, 2026, ISSN: 1566-2535. DOI: `https://doi.org/10.1016/j.inffus.2025.103599`. Accessed: Sep. 5, 2025.

[27] P. Belcak et al., “Small Language Models are the Future of Agentic AI,” arXiv preprint arXiv:2506.02153, 2025. Accessed: Sep. 26, 2025.

[28] C. Stryker. “What is tool calling?” Accessed: Sep. 25, 2025. [Online]. Available: `https://www.ibm.com/think/topics/tool-calling`.

[29] Microsoft. “Tools,” Accessed: Oct. 2, 2025. [Online]. Available: `https://microsoft.github.io/autogen/stable/user-guide/core-user-guide/components/tools.html`.

[30] Cursor. “Tool Calling,” Accessed: Sep. 25, 2025. [Online]. Available: `https://cursor.com/learn/tool-calling`.

[31] Anthropic. “Introducing the Model Context Protocol,” Accessed: Sep. 26, 2025. [Online]. Available: `https://www.anthropic.com/news/model-context-protocol`.

[32] Google. "What is the MCP and how does it work?" Accessed: Sep. 26, 2025. [Online]. Available: `https://cloud.google.com/discover/what-is-model-context-protocol?hl=en`.

[33] Model Context Protocol. "Architecture overview," Accessed: Oct. 2, 2025. [Online]. Available: `https://modelcontextprotocol.io/docs/learn/architecture`.

[34] Model Context Protocol. "Understanding MCP servers," Accessed: Oct. 2, 2025. [Online]. Available: `https://modelcontextprotocol.io/docs/learn/server-concepts`.

[35] Microsoft. "AutoGen - Open-Source Framework for Agentic AI," Accessed: Sep. 26, 2025. [Online]. Available: `https://www.microsoft.com/en-us/research/project/autogen/`.

[36] Microsoft. "API Reference," Accessed: Sep. 26, 2025. [Online]. Available: `https://microsoft.github.io/autogen/stable/reference/index.html`.

[37] Q. Wu et al., "Autogen: Enabling next-gen llm applications via multi-agent conversation," in COLM 2024, Aug. 2024.

[38] Microsoft. "Human-in-the-Loop," Accessed: Sep. 26, 2025. [Online]. Available: `https://microsoft.github.io/autogen/stable/user-guide/agentchat-user-guide/tutorial/human-in-the-loop.html`.

[39] A. Fourney et al., "Magentic-one: A generalist multi-agent system for solving complex tasks," arXiv preprint arXiv:2411.04468, 2024. Accessed: Sep. 24, 2025.

[40] Google. "From data warehouse to autonomous data and AI platform," Accessed: Sep. 26, 2025. [Online]. Available: `https://cloud.google.com/bigquery?hl=en`.

[41] Google. "BigQuery Overview," Accessed: Sep. 26, 2025. [Online]. Available: `https://cloud.google.com/bigquery/docs/introduction`.

[42] Google. "BigQuery API Client Libraries," Accessed: Sep. 26, 2025. [Online]. Available: `https://cloud.google.com/bigquery/docs/reference/libraries#client-libraries-install-python`.

[43] Google. "Agent Development Kit," Accessed: Sep. 26, 2025. [Online]. Available: `https://google.github.io/adk-docs/`.

[44] Google. “MCP Toolbox for Databases,” Accessed: Sep. 26, 2025. [Online]. Available: `https://googleapis.github.io/genai-toolbox/resources/tools/bigquery/`.

[45] The Elixir Team. “Kernel,” Accessed: Sep. 29, 2025. [Online]. Available: `https://hexdocs.pm/elixir/main/Kernel.html#%7C%3E/2`.

[46] Google. “Introduction to SQL in BigQuery,” Accessed: Sep. 24, 2025. [Online]. Available: `https://cloud.google.com/bigquery/docs/introduction-sql`.

[47] Google. “UNNEST operator,” Accessed: Sep. 26, 2025. [Online]. Available: `https://cloud.google.com/bigquery/docs/reference/standard-sql/query-syntax#unnest_operator`.

[48] Google. “Welcome to Looker Studio,” Accessed: Sep. 26, 2025. [Online]. Available: `https://cloud.google.com/looker/docs/studio`.

[49] Google. “Connect to Google BigQuery,” Accessed: Sep. 26, 2025. [Online]. Available: `https://cloud.google.com/looker/docs/studio/connect-to-google-bigquery`.

[50] R. Fitzpatrick, The Mom Test. CreateSpace Independent Publishing Platform, 2013, ISBN: 9781492180746.

[51] S. Lauesen, Software requirements: styles and techniques. Pearson Education, 2002, ISBN: 0 201 74570 4.

[52] P. Jalote, A concise introduction to software engineering. Springer, 2008, ISBN: 978-1-84800-301-9. DOI: `10.1007/978-1-84800-302-6`.

[53] Google. “Authenticate to BigQuery,” Accessed: Oct. 3, 2025. [Online]. Available: `https://cloud.google.com/bigquery/docs/authentication/#gcloud`.

[54] J. Becker, N. Rush, E. Barnes, and D. Rein, “Measuring the impact of early-2025 ai on experienced open-source developer productivity,” arXiv preprint arXiv:2507.09089, 2025. [Online]. Available: `https://arxiv.org/abs/2507.09089`.

[55] S. Henry. “Microsoft’s Agentic AI Frameworks: AutoGen and Semantic Kernel,” Accessed: Oct. 10, 2025. [Online]. Available: `https://devblogs.microsoft.com/semantic-kernel/microsofts-agentic-ai-frameworks-autogen-and-semantic-kernel/`.

[56] S. Henry, F. Niedtner, and S. Lagerkrans-Pandey. “AutoGen and Semantic Kernel, Part 2,” Accessed: Oct. 10, 2025. [Online]. Available: `https://devblogs.microsoft.com/semantic-kernel/semantic-kernel-and-autogen-part-2/`.

[57] Microsoft. “Microsoft Agent Framework,” Accessed: Oct. 10, 2025. [Online]. Available: `https://learn.microsoft.com/en-us/agent-framework/overview/agent-framework-overview`.

[58] Microsoft. “Microsoft Agent Framework,” Accessed: Oct. 10, 2025. [Online]. Available: `https://learn.microsoft.com/en-us/agent-framework/migration-guide/from-autogen/`.

[59] Google. “Write better log queries, faster: Introducing pipe syntax in BigQuery and Cloud Logging,” Accessed: Nov. 11, 2025. [Online]. Available: `https://cloud.google.com/blog/products/data-analytics/introducing-pipe-syntax-in-bigquery-and-cloud-logging`.

Appendix A

Interview questions

Category	**Question**
A1	What is your role - what are your tasks?
A2	How do you work with data analysis?
A2.3	What are the barriers to geting started with data analysis at the department?
A3	Is there anything in the work process or your tasks that you find frustrating?
A3.1	What would improve the experience of that task?
B1	How would describe your experience of using BigQuery?
B1.1	How often is there a need for you to create new queries, what is your process?
B1.2	How much time / effort does that usually take?
B1.3	In what ways do you use the retrieved data in your work?
B2	What is perceived as difficult or challenging with using BigQuery?
C1	How do you use, or do you plan to use AI tools in your work?

Label	**Explanation**
AX	Questions about general work and data analysis
BX	Questions about BigQuery and BigQuery usage
CX	Questions about AI usage

Appendix B

Prompt engineering

All prompts were made with the same user message, but with different prompt engineering-techniques. The queries all used the same LLM with the temperature set to 0, Top-P set to 0, and Top-K set to 5.

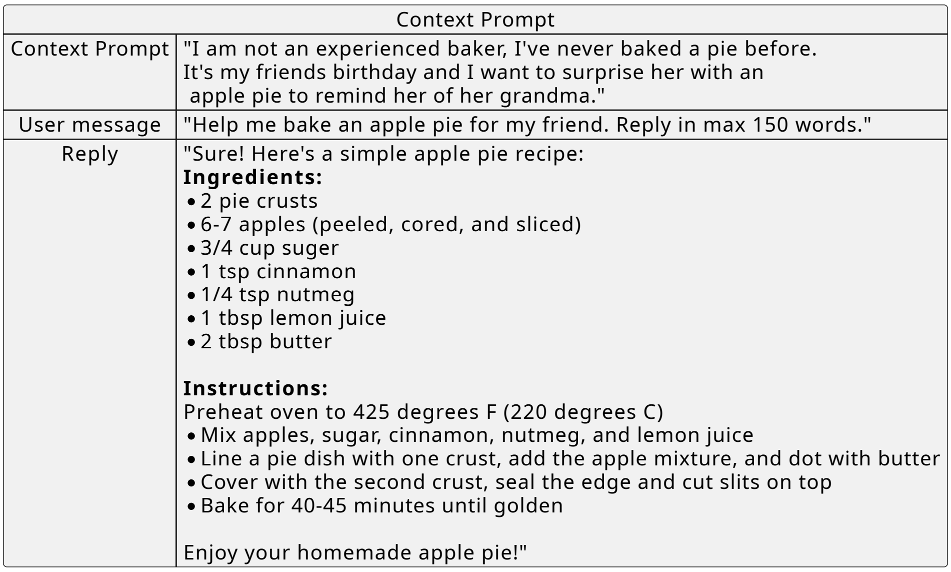
The different prompt engineering techniques can be seen in figures B.1, B.2, and B.3. Some differences can be seen, such as the system prompt following the instructions and listing the instructions as a numbered list. The role prompt, the experienced baker, suggesting an egg wash and cutting slits on top of the pie. The context prompt, the inexperienced user, suggests a higher temperature and has fewer instructions.

System Prompt	
System Prompt	"Reply first with a list of ingredients, their name and the amount. Then reply with numbered list for the instructions. Be concise and use common abbreviations where appropriate."
User message	"Help me bake an apple pie for my friend. Reply in max 150 words."
Reply	"**Apple Pie Recipe** Ingredients: • Pie crust (2 rounds) • Apples (6-7, peeled & slices) • Sugar (3/4 cup) • Brown suger (1/4 cup) • Cinnamon (1 tsp) • Nutmeg (1/4 tsp) • Lemon juice (1 tbsp) • Butter (2 tbsp) Instructions: 1. Preheat oven to 375 degrees F (190 degrees C) 2. Mix apples, sugars, cinnamon, nutmeg, and lemon juice. 3. Line pie dish with one crust, add apple mixture. 4. Dot with butter, top with second crust. Seal edges. 5. Cut slits in top crust, bake for 45-50 mins until golden. 6. Cool beore serving Enjoy your pie!"

Figure B.1: Utilisation of a system prompt tp instruct the LLM to format the output in a certain manner, ingredients first and then numbered instructions.

Role Prompt	
Role Prompt	"You are a professional baker. /With Years of experience working in bakeries and as a culinary teacher. You are known for being helpful to fellow bakers."
User message	"Help me bake an apple pie for my friend. Reply in max 150 words."
Reply	"Sure! Here's a simple recipe for a classic apple pie! **Ingredients:** • 2 pie crusts • 6-7 apples (peeled, cored, and sliced) • 3/4 cup suger • 2 tbsp flour • 1 tsp cinnamon • 1/4 tsp nutmeg • 1 tbsp lemon juice • 1 egg (for egg wash) **Instructions:** • Preheat your oven to 375 degrees F (190 degrees C) • Mix apples with sugar, flour, cinnamon, nutmeg, and lemon juice • Line a pie dish with one crust, add the apple mixture • Cover with the second crust, seal the edges, and cut slits on top • Brush with egg wash for a golden finish • Bake for 45-50 minutes until golden brown • Let it cool before serving. Enjoy your homemade apple pie!"

Figure B.2: A role prompt used to provide the LLM with a role as an experienced baker that has worked as an culinary teacher and is helpful to other bakers.



Context Prompt	
Context Prompt	"I am not an experienced baker, I've never baked a pie before. It's my friends birthday and I want to surprise her with an apple pie to remind her of her grandma."
User message	"Help me bake an apple pie for my friend. Reply in max 150 words."
Reply	"Sure! Here's a simple apple pie recipe: **Ingredients:** • 2 pie crusts • 6-7 apples (peeled, cored, and sliced) • 3/4 cup suger • 1 tsp cinnamon • 1/4 tsp nutmeg • 1 tbsp lemon juice • 2 tbsp butter **Instructions:** Preheat oven to 425 degrees F (220 degrees C) • Mix apples, sugar, cinnamon, nutmeg, and lemon juice • Line a pie dish with one crust, add the apple mixture, and dot with butter • Cover with the second crust, seal the edge and cut slits on top • Bake for 40-45 minutes until golden Enjoy your homemade apple pie!"

Figure B.3: A context prompt used to provide the LLM with background information regarding the users previous baking experience.

Appendix C

Structured and unstructured output

Output can be structured and unstructured. With structured output (figure C.2), the output can be tailored to the applications needs. In this example: the output is formatted to JSON with one part for the LLMs text reply and one part for the SQL query that the LLM generates. Compared to the unstructured output (figure C.1), where the parts are intertwined and it's harder to extract the SQL-query from the reply.

Other than changing a parameter to enable structured output and adding a field for structured output description, the two examples were set up in the same manner, with identical prompting techniques and settings.

```
-- TextMessage (user) --
Write me a SQL query for the DogProfiles table to select the
    average dog age.
-- ModelClientStreamingChunkEvent (AI Agent) --
Thank you for providing the database description. I’ll now
    write a SQL query to select the average
dog age from the DogProfiles table. I’ll calculate the age
    from the dogDateOfBirth field.


-- Query to calculate the average age of dogs from the
    DogProfiles table
-- We first convert the unix timestamp to a date, then
    calculate the age in years
-- Finally, we take the average of these ages
‘‘‘sql
WITH DogAges AS (
  SELECT
    dogId,
    dogName,
    -- Calculate age in years from dogDateOfBirth (unix
        timestamp)
    TIMESTAMP_SECONDS(dogDateOfBirth) AS dogBirthDate,
    DATE_DIFF(CURRENT_DATE(), DATE(TIMESTAMP_SECONDS(
        dogDateOfBirth)), YEAR) AS age
  FROM
    DogProfiles
)

SELECT
  AVG(age) AS average_dog_age
FROM
  DogAges;
‘‘‘

TERMINATE
== Summary ==
Stop reason: Text ’TERMINATE’ mentioned
```

Figure C.1: Agent reply without structured output. The generated SQL is intertwined with normal text reply and is difficult to extract.

```
-- TextMessage (user) --
Write me a SQL query for the DogProfiles table to select the
    average dog age.
-- ModelClientStreamingChunkEvent (AI Agent) --
{"text_reply": "I will write a query to select the average dog
     age from the DogProfiles table. Since the dogDateOfBirth
    is stored as a Unix timestamp, I will convert it to a date
    format and then calculate the dog’s age in years. TERMINATE
    ", "sql_reply": "-- Query to select the average dog age
    from the DogProfiles table\nSELECT\n  AVG(dog_age) AS
    average_dog_age\nFROM (\n  SELECT\n    TIMESTAMP_SECONDS(
    dogDateOfBirth) AS dog_birth_date,\n    -- Calculate dog’s
    age in years\n    DATE_DIFF(CURRENT_DATE(), DATE(
    TIMESTAMP_SECONDS(dogDateOfBirth)), YEAR) AS dog_age\n
    FROM\n    DogProfiles\n);"}
== Summary ==
Stop reason: Text ’TERMINATE’ mentioned
SQL:
-- Query to select the average dog age from the DogProfiles
    table
SELECT
  AVG(dog_age) AS average_dog_age
FROM (
  SELECT
    TIMESTAMP_SECONDS(dogDateOfBirth) AS dog_birth_date,
    -- Calculate dog’s age in years
    DATE_DIFF(CURRENT_DATE(), DATE(TIMESTAMP_SECONDS(
        dogDateOfBirth)), YEAR) AS dog_age
  FROM
    DogProfiles
);
```

Figure C.2: Agent output when asked to create a SQL query after structured output was enabled. The generated SQL can be extracted when parsing the output as JSON.

Appendix D

Code examples: Agent

Code snippets for the agent functionality. Configuring the model that powers the agent, constructing the agent, setting up structred output, and defining a tool.

```
from autogen_core.models import ChatCompletionClient
from autogen_ext.models.openai import
    OpenAIChatCompletionClient

model_client = OpenAIChatCompletionClient(
    model = model,
    base_url = base_url,
    api_key = api_key,
    temperature = 0.0,
    seed = 2277,
    top_p = 1.0,
    model_info = {
        "structured_output": True,
        "family": "unknown",
        "vision": False,
        "json_output": True,
        "function_calling": True,
    },
)
```

Figure D.1: A model client defined for a model with OpenAI-like API functionality. Parameters for seed, temperature, and top_p can be altered to affect LLM output.

```
from autogen_agentchat.agents import AssistantAgent

sql_writer = AssistantAgent(
    name="SQL_Expert",
    model_client=model_client,
    system_message="""
    You are an expert in SQL.
    You help the user construct SQL queries that
    are pragmatic and well documented. Use your
    databaseSchemaTool to view the schema of the
    database.
    """,
    tools=[tools.databaseSchemaTool],
    output_content_type=AgentResponse,
    model_client_stream=False,
    reflect_on_tool_use=True,
)
```

Figure D.2: An AssistantAgent being defined with a specified model client, a system prompt, tools that the agent can access, and potential custom output structure.

```
from pydantic import BaseModel

class AgentResponse(BaseModel):
    text_reply: str
    sql_reply: str
```

Figure D.3: The structured output used for the sql_writer-agent. Messages are divied into two parts in JSON, a text_reply containing normal text and a sql_reply containing only generated SQL and SQL comments.

```
assert isinstance(result.messages[-1], StructuredMessage)
assert isinstance(result.messages[-1].content, AgentResponse)
print("SQL: \n", result.messages[-1].content.sql_reply)
```

Figure D.4: Extracting and printing the generated SQL from the structured output from the sql_writer agent.

```
from autogen_core tools import FunctionTool

async def getDatabaseSchema():
    schema = utility.load_database_schema() // Helper function to load the JSON schema as a Python dict
    return schema


databaseSchemaTool = FunctionTool(getDatabaseSchema,
                    description="Database schema in dict with column names, types, mode, and descriptor",
                    strict=True)
```

Figure D.5: A tool definition for retrieving the database schema as a Python dictionary

Appendix E

Log example

Example of a log output for a simple query with a single agent. The user and user settings are logged, together with the initial prompt and the amount of time it took to run the query and all messages generated for the prompt

```
"timestamp": "2025-10-27_16:09:01",
"user": "andrk",
"config": {
  "model": {
    "name": "local_model",
    "api_key": "",
    "base_url": "localhost:2277"
  },
  "settings": {
    "use_reviewer": false,
    "save_logs": true,
    "auto_execute_queries": false
  }
},
"prompt": "List the amount of iOS ( iPhone ) users as a percentage of all users",
"exec_time": 11.77,
"using_tui": true,
"messages": {
  "msg-0": {
    "agent": "sql_writer",
    "text": "TERMINATE",
    "sql": "-- This query calculates the percentage of iOS users (iPhone) out of the total users.\nWITH user_counts AS (\n  SELECT\n    COUNT(DISTINCT CASE WHEN device.mobile_model_name LIKE ’%iPhone%’ THEN user_pseudo_id END) AS ios_user_count,\n    COUNT(DISTINCT user_pseudo_id) AS total_user_count\n  FROM\n    ‘companion_*‘\n  WHERE\n    _TABLE_SUFFIX BETWEEN FORMAT_DATE(\"%Y%m%d\", DATE_SUB(CURRENT_DATE(), INTERVAL 14 DAY))\n    AND FORMAT_DATE(\"%Y%m%d\", DATE_SUB(CURRENT_DATE(), INTERVAL 0 DAY))\n)\nSELECT\n  ios_user_count,\n  total_user_count,\n  ROUND((ios_user_count / total_user_count) * 100, 2) AS ios_user_percentage\nFROM\n  user_counts;"
  }
}
```

Figure E.1: Logged entry using the custom logging structure when using the prototype.