Assessing Continuity in Agentic Development Across Multiple Development Cycles

Teo Jonsäter

Anna Svensson

Department of Electrical and Information Technology
Lund University

Supervisor: Christin Swahn

Examiner: Christian Nyberg

June 1, 2026

Printed in Sweden
Tryckeriet i E-huset, Lund

Abstract

Large Language Models (LLMs) are increasingly being used as a key technology in agent-based systems for software-related tasks. These systems are intended to support long-term tasks and may consist of multiple interacting agents working towards shared goals. In these scenarios, agents are expected to work on the same codebase over multiple sessions and development cycles. In long-term software development tasks, maintaining continuity across multiple development cycles remains a significant challenge. Due to limited context windows, agents often treat development sessions independently, leading to a loss of understanding in important architectural decisions over time.

This thesis evaluates five combinations of context engineering strategies across two development cycles in order to analyze how differing strategies affect the agent's continuity. These strategies were tested using a mock web development project, where the agent was tasked with initializing and further developing the codebase while adhering to specific architectural and stylistic decisions (referred to as mid-session decisions) made in previous development cycles. The thesis benchmarks the effectiveness of these strategies based on requirement compliance, decision consistency, time- and token efficiency.

The results indicate that implementing any form of persistent context management significantly improves the agent's ability to adhere to mid-session decisions compared to a baseline of no strategies. Among the tested methods, decision 2 (decision log) proved to be the most effective in maintaining continuity, resulting in the highest decision consistency while remaining cost-effective and time-efficient. This finding suggests that for sustainable agentic development, context management strategies should prioritize the preservation of specific architectural decisions over general documentation of finished tasks.

Keywords: Agentic development, AI, Context management, Context strategies, Continuity.

Sammanfattning

Stora språkmodeller (LLM:er) används i allt större utsträckning som en viktig teknologi i agentbaserade system för mjukvarurelaterade uppgifter. Dessa system är avsedda för att kunna hantera långsiktiga uppgifter och kan bestå av ett flertal interagerande agenter som arbetar mot gemensamma mål. I dessa scenarier förväntas agenterna arbeta med samma kodbas över flera sessioner och utvecklingscykler. I långvariga mjukvaruutvecklingsprojekt är det fortfarande en stor utmaning att upprätthålla kontinuiteten över flera utvecklingscykler. På grund av begränsade kontextfönster så behandlar agenterna ofta utvecklingssessionerna som fristående. Detta leder med tiden till att förståelsen för viktiga arkitektoniska beslut riskerar att glömmas bort.

Denna rapport utvärderar fem kombinationer av strategier för kontexthantering över två utvecklingscykler, i syfte att analysera hur olika strategier påverkar agentens kontinuitet. Dessa strategier testades genom att använda ett mock webbutvecklings projekt, där agenten hade i uppgift att initiera och vidareutveckla kodbasen, samtidigt som den skulle följa specifika arkitektoniska beslut (benämnt som sessionsbeslut) som fattats i tidigare sessioner och utvecklingscyklar. I rapporten jämförs strategiernas effektivitet utifrån upprätthållning av krav och sessionsbeslut, samt tid- och kostnadseffektivitet.

Resultaten visar att agentens förmåga att följa sessionsbeslut ökar markant när någon form av kontexthanteringsstrategi implementeras, jämfört med en utgångspunkt utan strategier. Bland de prövade metoderna så visade sig strategi 2 (beslutslogg) vara den mest effektiva strategin för att upprätthålla kontinuitet. Denna strategin resulterade i den högsta upprätthållningen av sessionsbeslut, samtidigt som den förblivit kostnads- och tidseffektiv. Detta resultatet tyder på att, för hållbar agentisk utveckling, så borde kontexthanteringsstrategier prioritera att bevara specifika arkitektoniska beslut över generell dokumentation av avslutade uppgifter.

Nyckelord: Agentisk utveckling, AI, Kontexthantering, Kontextstrategier, Kontinuitet.

Preface

We would like to thank our supervisors at Bosch R&D Center Lund, Adam Koch, Staffan Lindgren and Samuel Peltomaa, without whom this thesis would not be possible. A special thanks to Samuel Peltomaa for being extra committed and supportive throughout this thesis and helping us brainstorm and coming up with different ideas.

We would also like to thank our supervisor Christin Swahn for being supportive and helping us navigate the strict world of academic writing, and whose regular supervisory meetings always made sure that we stayed on track.

Teo Jonsäter & Anna Svensson

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

This chapter introduces the problem that occurs during development with agentic AI systems in regards to continuity and context. It proposes a solution based on a few context engineering strategies.

1.1 Background

Large Language Models (LLMs) are increasingly being used as a key technology in agent-based systems for software-related tasks. These systems are intended to support long-term tasks and may consist of multiple interacting agents working toward shared goals.

Robert Bosch AB is an international engineering and technology company that was founded in Germany in 1886. As an engineering-focused company, Bosch aims to be at the forefront when it comes to technological advancements. At Bosch R&D Center Lund, agentic AI is used in continuous software development scenarios within areas such as mobility, IoT, and embedded systems.

This is done with a multi-agent factory that uses a pipeline of agents, with differing specialties that feed into each other. These agents are all interacted with in user-sessions, where the user can discuss and create products together with the agents. See figure 1.1. In these scenarios, agents are expected to work on the same codebase over multiple sessions, often spanning days or weeks.

When working in a software development project, the full development process is divided into several key steps, following the SDLC - Software Development Cycle (see figure 1.2). When following the SDLC during agentic development, context needs to be managed at all stages, see figure 1.3. The challenge the AI team at Bosch R&D Center Lund is facing, appears when an already existing project is in the maintenance phase and needs to be further developed. During further development, the SDLC begins anew and the agents need to remember context from previous cycles.

Agents operate within a limited context window, which defines the maximum amount of information that can be provided to the model at any given time. Due to this limitation, agents typically treat each session independently, with a restricted amount of prior information being available when a new session begins. When these agents are used in long-running development tasks, several key challenges appear if context isn't managed properly:

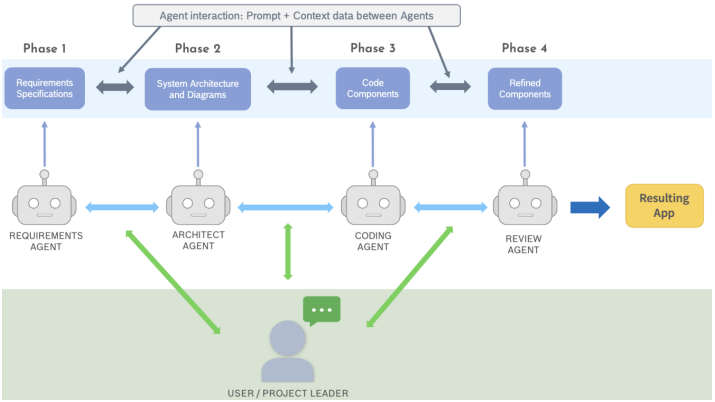

Figure 1.1: Multi-agent factory pipeline [1]

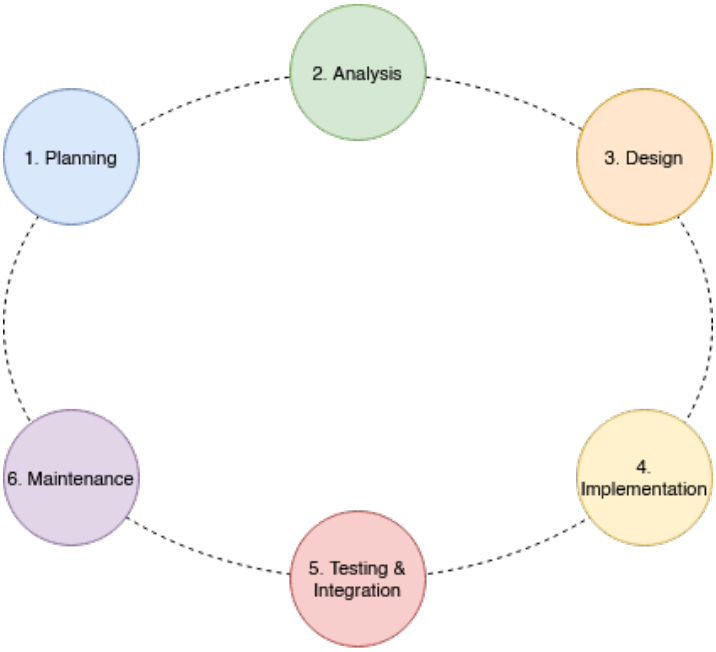

Figure 1.2: Software Development Cycle

- **Alignment:** Making sure that outputs align with documents and specifications.
- **Coherence:** Inconsistent or contradictory outputs.
- **Continuity:** Failure in maintaining a consistent understanding of previous decisions and/or constraints.
- **Quality:** Suboptimal design choices, duplicated work, or incorrect assumptions.

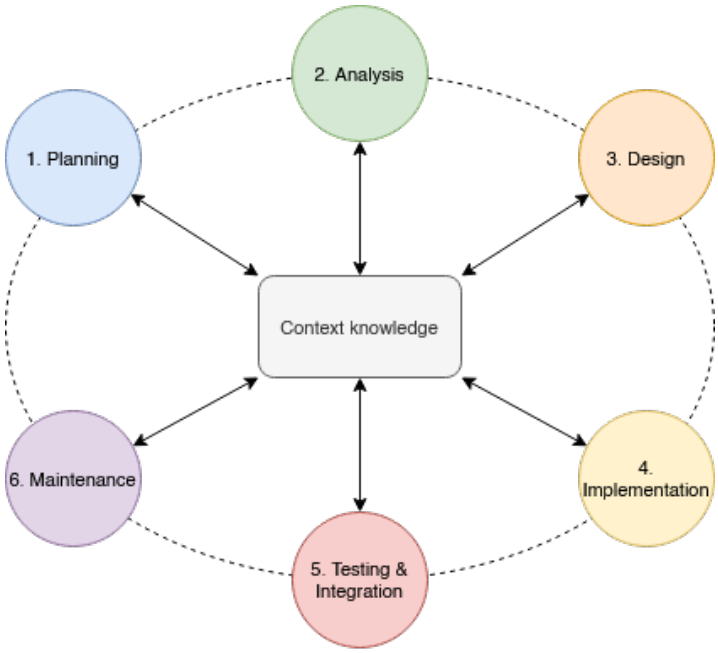

Figure 1.3: Software Development Cycle with context management

- **Token efficiency:** Excessive context may increase token usage and computational cost.

Current approaches can support individual tasks, but they provide limited support for maintaining long-term continuity across multiple cycles. An approach for managing these challenges is systematic context management. This means strategies and mechanics used to select, structure, retrieve, and store information that is provided and outputted to/from an agent throughout its operation. Context management can be viewed across the agent's lifecycle stages:

- **Pre-session stage:** Initializing the agent with relevant background information before a session begins.
- **Intra-session stage:** Allowing the agent to dynamically retrieve and update context while performing tasks.
- **Post-session stage:** Storing and summarizing information from a completed session for use in future sessions.

The solution this thesis proposes is able to support long-term agent behavior by storing essential knowledge across sessions while adhering to context window limitations.

In order to determine which context strategies give the best results, the results need to be analyzed and benchmarked. The method used for benchmarking the result in this thesis is with a mock project. With a specified mock project, the agent factory can be used to further develop the existing project with new features.

This will be done with different context strategies, and the results can be analyzed and ranked.

1.2 Purpose

The purpose of the thesis is to examine how improved context management can simplify the development cycle when already established agent-created products are to be further developed. The expected result is to find a strategy for managing multiple agent sessions in which agents work on the same code base throughout the development cycle, while still maintaining continuity.

1.3 Goals

The goals of this thesis are the following:

- Specify methods on how to benchmark the results produced on the basis of continuity.
- Specify a mock project for an agent system to design and develop.
- Investigate which context management strategies, for agentic use over multiple tasks and sessions, get the best benchmark results.

1.4 Problem statement

Throughout this thesis, the following questions will be answered:

1. How is context management currently handled in continuous software development tasks in two already established agentic softwares?
2. Which pre-session context management strategies are suitable for initializing AI agents?
3. Which intra-session context management strategies are suitable for dynamic context retrieval?
4. What post-session context management strategies are suitable for summarization, memory extraction, and persistence?
5. What evaluation criteria and metrics are appropriate to assess continuity?

1.5 Motivation of thesis

At Bosch, multi-agent systems are already integrated into the software development process. However, as development progresses into later stages of the development cycle, particularly during the maintenance phase, maintaining continuity of context across development cycles becomes increasingly challenging. Addressing these challenges is essential to ensure long-term efficiency in agent-supported development workflows.

By focusing on how contextual information can be stored and reused in subsequent development cycles, the thesis aims to contribute to a more sustainable and maintainable agent-based development processes.

From a broader perspective, artificial intelligence represents a strategically important and rapidly evolving technological field. Strengthening competence in agentic development practices supports long-term innovation and competitiveness. Therefore, this thesis contributes not only to Bosch's internal development processes, but also to the advancement of applied AI research in Sweden.

1.6 Limitations

This thesis does not include the development of agentic AI systems. Bosch currently provides an existing AI agent system based on prompt-driven interaction, which will be used solely for generating the mock project.

Regarding challenges related to context management, the scope of the thesis is limited to improvements in continuity. Issues such as alignment, coherence, quality, and token efficiency are therefore considered out of scope.

Furthermore, the analysis of different context management strategies does not address context sharing between multiple agents, as this functionality is already sufficiently handled by Bosch's existing tools. Instead, the focus lies on how contextual information from an entire development cycle can be stored and subsequently reused in a new development cycle.

1.7 Division of work

In table 1.1 the division of work between the authors can be seen for this thesis.

Table 1.1: Division of work between the authors of the thesis

	Anna Svensson	Teo Jonsäter
Analysis	50 %	50 %
Design	60 %	40 %
Implementation	40 %	60 %
Testing/Evaluation	50 %	50 %
Report	50 %	50 %
Presentation	60 %	40 %
Poster	40 %	60 %

Chapter 2
Technical Background

This chapter introduces the technical tools and concepts that were used in the development of this thesis. It provides a basic understanding of the area.

2.1 Artificial Intelligence

Artificial Intelligence (AI) is an umbrella term that, in general terms, describes systems that can think and learn like a human, i.e., solve complex tasks that normally require human reasoning and decision making [2]. AI is usually divided into multiple fields of concept, but most importantly:

- Machine learning
- Deep Learning

AI is an area that is currently being meticulously researched and developed, and its definitions and core methods may change drastically in the near future.

2.1.1 Machine Learning

Machine Learning is a type of AI that is a cornerstone to most modern AI systems. It works by analyzing massive datasets of human-made labeled training data, and makes predictions for future data, based on the patterns found in the training data [3].

2.1.2 Deep Learning

Deep learning is a type of machine learning that uses deep neural networks to identify intricate patterns and nuances from complex data. Unlike traditional machine learning, deep learning can automatically extract features directly from raw data, making it less dependent on large labeled datasets [3].

2.2 Natural Language Processing

Natural Language Processing (NLP) is a subfield of artificial intelligence that with the help of machine learning and deep learning can understand and communicate with text and data [4].

2.3 Large Language Model

LLM:s are language models that are built on Natural Language Processing (NLP) and machine learning. They belong to a category of deep learning models that have been trained on large amounts of data, and can use this data to learn and predict patterns in text, and use it to generate new text [5]. Due to the stochastic nature of LLM:s, the output may vary during repeated generation with the same input.

2.3.1 Prompting

When using a LLM the input to the LLM is called a prompt. When writing specific instructions for the LLM to understand, interpret and respond to, that is called prompting [6].

The way a prompt is worded can make a lot of difference on the output. Small changes can lead to drastically different outputs. The technique of adjusting and writing the perfect prompt is called prompt engineering [6].

2.3.2 Tokens

Tokens are the units of data that LLM:s are based on, they can be words, characters or a combination of words and punctuation. What counts as a token depends on the LLM. Tokens are generated by LLM:s when they decompose text, and are then analyzed in order to provide semantic relationships between tokens. The LLM uses these relationships to generate sequences of tokens as outputs [7].

2.3.3 Context

Context defines the knowledge that an LLM has available and can use to base their output on [8]. With insufficient context, a model may "hallucinate" results that are inaccurate and not based on training data [9].

Context is measured in tokens. The LLM cannot consider an infinite amount of context, and is limited by the so called "context window". The larger the context window is, the more information can be inputted and the more accurate outputs can be given with less hallucinations [10]. The amount of context given does however not always correlate to an increase in output accuracy. When the number of tokens increases, the ability for the LLM to use the context decreases. This concept is called context rot [8]. There are multiple strategies to finding the optimal set of tokens for the LLMs context window so that it can have high performance and still give accurate and correct answers. During a session, the context grows and it is important to understand what and how much should be saved in the context window. These strategies are referred to as context engineering [8].

By providing specified knowledge to an agent in the form of context, key challenges such as continuity can be improved. This is an area that will be explored in this thesis.

2.4 Agents

AI agents are systems that can make decisions, complete tasks, and interact with external tools and environments. They are LLM-powered to help understand inputs from users and to generate answers, both text and code. Agents use context to help remember what has been done and what is to be done. To achieve complex tasks, agents use tool calls to interact with the environment and execute actions [11], [12].

Agents such as Codex will be used in this thesis in order to evaluate and examine different context engineering strategies.

2.4.1 Codex

Codex is an AI Agent developed by OpenAI. Codex is specialized in software development and can work on many different tasks, such as coding features, explaining a codebase and fixing bugs [13].

Codex can be used in several different environments, such as a dedicated app, in an IDE, in a terminal (CLI), and a cloud version in the browser [14].

In order to understand a project better, Codex uses a system called AGENTS.md to start each session with consistent expectations and better understanding of project structure and routines [15]. AGENTS.md can be described as a README for agents, it's a dedicated file where the user can provide context and instructions to help the agent navigate the project. They can consist of quickstarts, descriptions, instructions and project structure [16].

2.5 NotebookLM

NotebookLM is an AI-powered research tool created by Google that allows for note-taking and source summation based on user-provided documents and sources. By analyzing provided sources, NotebookLM can extract key concepts and present them to the user in a structured format where the output is based on the sources with citations [17].

2.6 React

React is a JavaScript library for creating user interfaces with interactions and state management. It allows for component building and interactivity in a front-end environment. React by default uses JavaScript with `.jsx` files (`.tsx` if using TypeScript) to indicate components. It can also be configured to use TypeScript [18].

2.7 Vite

Vite is an open source frontend build tool that uses JavaScript modules and browser APIs to build web applications. It supports a number of different frameworks, such as React. Vite provides features to package and run source code, a

development server for local development, and a build command to deploy the app to a production server [19]. Using Vite is the recommended method for creating a React application [20].

2.8 TypeScript

TypeScript is a programming language that builds on JavaScript, adding additional syntax to provide JavaScript with strong typing. TypeScript compiles into JavaScript, allowing it to run on all web environments where JavaScript is implemented [21].

2.9 Tailwind

Tailwind is a CSS framework that, unlike other frameworks, does not provide predefined classes for elements. Tailwind instead supplies a list of utility CSS classes that can be mixed and matched to style individual elements, without having to write manual CSS code [22].

2.10 Recharts

Recharts is a component library for React that provides a library of customizable charts that can be used in a web app [23].

Chapter 3
Method and analysis

This chapter outlines the method for the thesis. It describes the different phases that the work divided into, and what was done in each phase. It describes what the mock project is, how it was produced and why. The chapter also details the methods used for producing generation prompts, mid-session decisions, which strategies to test and how to test them.

3.1 Work phases

The work on the thesis is divided into 4 phases that focus on completing different aspects of the thesis. These phases are based on a GANTT-scheme (see figure 3.1) that was made at the beginning of the thesis with regard to deadlines and exam periods. Some adjustments were made to the plan due to delays. The phases were executed sequentially.

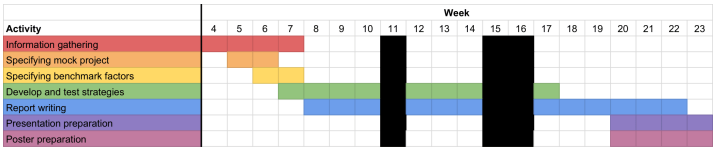

Figure 3.1: Initial GANTT-scheme

3.1.1 Information gathering phase

The first phase of the thesis was information gathering. A large part of this phase was spent on gathering key information about the technologies that were going to be used in the thesis. Information such as how agents work, criteria that could be used to judge an LLMs output, and what techniques that could be used to generate code with agents. This was done by searching for keywords on Google and using chatbots such as ChatGPT and Gemini to gather relevant sources.

Within this phase the scope of the thesis was also narrowed down to fit the needed time frame and level of complexity.

3.1.2 Specification phase

The second phase was the specification phase, which was mostly used to specify details about the mock project. The mock project is further specified in chapter 3.2. Key details such as requirements specifications, features, and the prompts required to generate the mock project were all specified during this phase.

There was also a lot of focus spent on deciding which context management strategies were the most suitable for the mock project, as well as specifying how to test and evaluate the strategies. These decision suggestions were presented by the thesis authors and decided on during consultation with the AI team at Bosch R&D Center Lund. For more details about the strategies, see chapter 3.3.

3.1.3 Development phase

The third phase was the development phase. This is the phase where data from the tests were collected and compiled into metrics and tables. The mock project was developed three times for each combination of context strategies and the agent was tested on different aspects on each one of the generated mock projects.

3.1.4 Evaluation phase

The fourth and final phase was the evaluation phase were the results from the development phase were evaluated and analyzed to see which combination of strategies work the best for improving continuity.

3.2 Mock project as a testing environment

A method of evaluating the output of an agent is to give it a specific task to perform and evaluate the quality through different metrics [24]. Due to the nature of this thesis, and based on suggestions from the AI engineering team at Bosch R&D Center Lund, the task was specified as a mock project. The mock project would consist of a couple of features that could be repeatedly generated by a coding agent.

In order to evaluate the agents output based on continuity over multiple development cycles, the mock project needed to be built over several sessions and at least two development cycles. In this case, the mock project is a frontend-only LLM dashboard web app built in React that - based on mock data - can display LLM usage with statistics. The features and functionality of the mock project were decided based on the need from the AI engineering team at Bosch R&D Center Lund for an LLM-usage dashboard, and because of its relevant level of complexity. This allows for a project that can be easily generated by a coding agent, and that can also be used outside of the scope of this thesis.

The web app is built around UsageEvents, a type of object that represents one LLM-usage. By having a large supply of UsageEvents, the app can showcase key statistics about their usage and Key Performance Indicators (KPIs).

For full requirement specification of the mock project, see appendix A, figure A.2

3.2.1 Architecture and technical stack

To ensure usability and relevance, the web app is built with the following stack:

- **Framework:** React 18 with Vite and TypeScript for strict type safety.
- **Styling:** Tailwind CSS for UI construction.
- **Data layer:** A static JSON-based mock database in order to decrease complexity by not having to implement external API management.
- **Graphs:** The component-library Recharts for rendering complex time-series and breakdown charts

3.2.2 Functional features

The mock project has several different features over three pages; a dashboard page, an events page, and a comparison page.

Data Generation and validation

In order to populate the web app with mock data, there needed to be some sort of data generation. This data was then used throughout the whole web app. The web app contains a script that generates realistic mock data with configurable parameters. The parameters allows a user to change specific factors about generation such as number of events, time spread and users.

The web app performs validation on all UsageEvents and will consciously fail to load the web app if something is wrong with the data.

Filtering

Throughout all the web apps pages, there is a shared filter menu that can be set and applied to all instances where events are displayed (e.g. statistics, events table, etc). The user of the web app can filter on the following different factors:

- **Time range:** When was the usage performed?
- **User:** Who performed the usage?
- **LLM:** Which language model was used?
- **Token metric:** Input, output or total

Dashboard

The dashboard page has multiple features. The purpose of this view is to aggregate raw usage data into Key Performance Indicators (KPIs) and graphs. The dashboard showcases the following KPIs:

- Total cost
- Total tokens (input, output or total)
- Request count (number of events)

- Average cost per request
- Average tokens per request

The graphs showcasing on the page is a timeline chart with bucketing over cost and tokens, a breakdown chart over cost by model and a calendar heatmap showcasing the events per day over a selected year.

All data showcased on the dashboard is affected by the currently applied filters.

For an example of what the dashboard page could look like, see figure 3.2. Due to the stochastic nature of agentic code development, the exact design of this page can vary for every iteration. This image shows the page from one of the generated iterations.

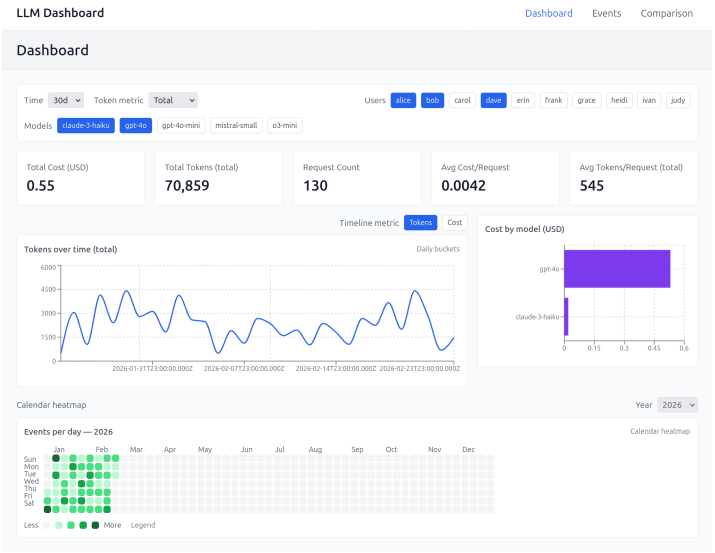

Figure 3.2: Mockup of dashboard page

Events visualization and export

The events page showcases a tabular view of the raw usage event data. It allows the user to sort the data based on the columns. Every event has a detailed modal view showcasing raw data when clicked. To increase performance, the page implements pagination showcasing 50 events per page. As with the dashboard, all data showcased in the table is affected by the currently applied filters.

The table with applied sorting and filter can be exported into a CSV format for further analysis and handling in external programs.

For an example of what the events page could look like, see figure 3.3. Due to the stochastic nature of agentic code development, the exact design of this page can vary for every iteration. This image shows the page from one of the generated iterations.

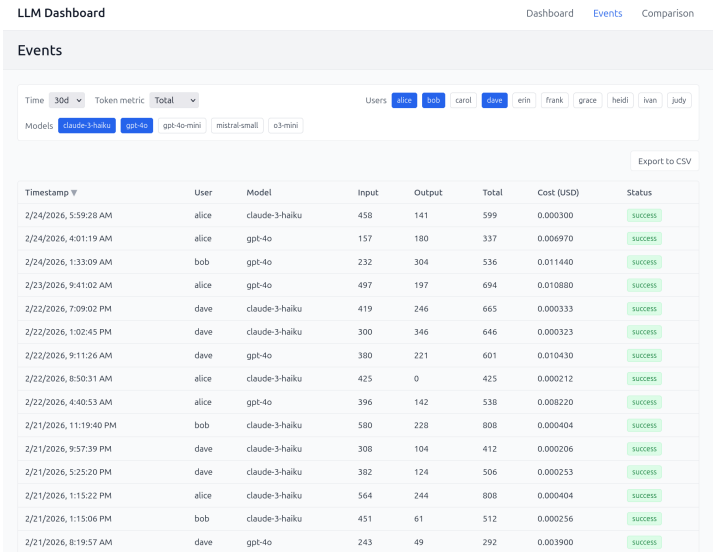

Figure 3.3: Mockup of events page

Comparative analysis

The comparison page allows the user to pick two different time periods and showcase the different KPIs from the dashboard page for the different time periods. The differences between the KPIs of the two periods are highlighted to showcase trends and divergence.

For an example of what the comparison page could look like, see figure 3.4. Due to the stochastic nature of agentic code development, the exact design of this page can vary for every iteration. This image shows the page from one of the generated iterations.

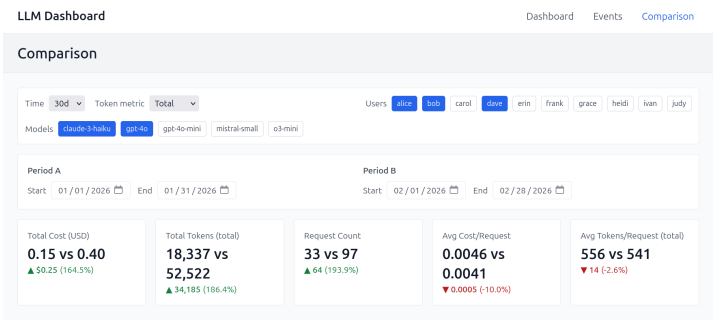

Figure 3.4: Mockup of comparison page

3.2.3 Development cycles

To enable assessment of continuity across multiple development cycles, the generative development of the mock project has been split up into two distinct development cycles, representing how a real development project might be structured.

- **Cycle 1:** Build up the first version of the mock project
- **Cycle 2:** Polish up already established features and implement new features

Cycle 1

The first cycle focused on building a release-ready version of the web app, that fulfilled the basic needs of an LLM-usage dashboard. For this cycle, a requirements specification document was produced containing all features and constraints. This initial version was made by an employee at the AI-team at Bosch R&D Center Lund, but was further modified by the authors to include continuity evaluation metrics. This first cycle included the following features:

- Basic outline of the mock project
- Event data model and data generation script
- Table showcasing all usage events
- Event filtering
- Usage KPIs on dashboard
- Breakdown and timeline charts for model, cost and tokens without a component library

For full requirement specification of cycle 1, see appendix A, figure A.1.

Cycle 2

After specifying features and requirements for cycle 1, the mock project was generated using the currently specified prompts without any context management techniques in order to produce an example version of the web app. This allowed for a visual example of what the web app should do and could look like. With this example project, it was easier to brainstorm different improvements and new features to be implemented in cycle 2. These improvements and new features were discussed and specified with the AI team at Bosch R&D Center Lund

This second cycle included the following features:

- Pagination on events table to increase performance
- Refactored all charts to use the Recharts library
- Added export functionality to the events table
- Comparison view for comparing KPIs across different time periods
- Calendar heatmap on dashboard for showcasing usage events per day

For full requirement specification of cycle 2, see appendix A, figure A.2.

3.2.4 Prompts

In order to generate each cycle for the mock project, a number of different prompts were produced across several sessions per cycle. Both development cycles were divided into several generative sessions in order to ensure that session context from previous prompts would not influence the impact of the context strategies.

The session breakdowns were based on features that the app needed, with one or two sessions per feature. Cycle 1 was broken down into the following sessions:

1. Scaffold + app shell
2. UsageEvent data model + loader + validation
3. Mock data generator
4. Events table + details
5. Filtering + aggregations + KPIs
6. Charts (timeline + breakdown)

Cycle 2 was broken down into the following sessions:

1. Refactor and improve existing features (charts library + pagination)
2. Export events table
3. Period comparison
4. Calendar heatmap

For a full session breakdown of each cycle with specific features, see appendix B figure B.1 and B.2.

Once all sessions were defined, it was time to produce the generation prompts. All prompts created for the web app base their structure and grammar on prompt engineering techniques in order to minimize hallucinations and inaccuracies [6]. More specifically, the following prompt engineering strategies were used:

- Contextual prompting
- Chain of thought
- One-shot

Based on these strategies, a first iteration of prompts were manually produced by the authors for cycle 1, resulting in a web app that followed almost all requirements. However, in order to see how different wordings and prompting strategies can affect the outcome, a second iteration was also produced. This second iteration of prompts used a new prompt engineering strategy called Automatic Prompt Engineering. This meant producing prompts using an LLM, in this case NotebookLM, with the prompt engineering guide [6] supplied as guidelines for generation. This produced a result that worked very similarly to the first iteration, but with some more requirement misses. It did however contain other features that were not thought of before, laying the ground work for some features that appear in cycle 2, such as pagination and a chart library. With both iterations not entirely fulfilling the requirements, a third iteration was made. This iteration combined

the strategies and structure from both previous attempts in order to produce a complete web app that matched all requirements needed for cycle 1. This third version became the base that cycle 2 would be building upon.

The first iteration of prompts for cycle 2 were created by using the same strategies and techniques as the third iteration of cycle 1. This resulted in a cycle 2 that fulfilled most requirements, but that still required some tweaking. After tweaks, a second iteration was created that matched all requirements needed for cycle 2.

With the prompts for cycle 1 and 2 produced, they were both slightly tweaked in order to ensure that they fit together, and that the first cycle did not unintentionally produce results that were to be implemented in cycle 2. This resulted in a final version of prompts for both cycles, which can be seen in appendix C figure C.1 and C.2. In these finalized prompts, there are also prompts for generating mid-session decisions, which can be further read in chapter 3.2.5.

3.2.5 Mid-session decisions

When testing continuity, the agent is tested on its ability to maintain consistent understanding of previous decisions and/or constraints. Therefore, a way to test this is by involving a number of mid-session decisions (sometimes referred to as simply "decision" in this thesis) that only appear in the prompts of a session and not in the requirement specification. This means that the agent is given extra rules to follow throughout the mock project but these rules are only mentioned mid session, meaning that it is up to the agent to remember it for later sessions. During evaluation and analysis of different strategies, decision adherence can be observed and analyzed.

When deciding on which mid-session decisions to implement in the mock project, it was important to make choices that would impact later sessions if not followed. By brainstorming with the AI team at Bosch R&D Center Lund the following mid-session decisions were specified:

1. Add the text "(Heading)" to each heading.
2. Add a comment at the top of each `.tsx` and `.ts` file with the time and date it was created.
3. Create a color palette for different LLM:s in the mock project that will be saved in a config-file, and later used everywhere a specific model is displayed somewhere.
4. When implementing a new page, its `.tsx` file should be placed in a pages folder.

After specifying these decisions, they were implemented into the prompts (see more about the prompts in chapter 3.2.4) at appropriate places, based on what the decision would affect. All the mid-session decisions were implemented in cycle 1 to allow for analysis of how well the decisions are followed in cycle 2.

The first mid-session decision was implemented in session 1 by adding another prompt after the main content prompt for that session. The prompt added can be seen in figure 3.5.

```
I want all headings in the app to have the text "(Heading)"
↪  appended at the end of the content of each heading tag.
↪  Example: <h1>Welcome (Heading)</h1>. This must be done
↪  manually for every heading, no automation
```

Figure 3.5: Prompt for mid-session decision 1

The second mid-session decision was implemented in session 2 before the main content prompt for that session. This position was chosen due to many new files being added later in the session. The prompt added can be seen in figure 3.6.

```
I want all `.tsx` and `.ts` files to have a comment at the top
↪  of the files containing the date and time the file was
↪  created. For already existing files, use the current date
↪  and time for created. This must be done manually, no
↪  automation.
```

Figure 3.6: Prompt for mid-session decision 2

The third mid-session decision was implemented in session 3 after the main content prompt for that session. This position was chosen due to the preceding prompt defining how many models that are to be used in the app. Therefore the mock data needs to be generated before this decision is implemented. The prompt added can be seen in figure 3.7.

```
I want every model used in the mock data to be displayed in the
↪  app with its own specified color. When the name of a model
↪  is displayed, it should be colored with its own color.

Pick a color palette where every model is represented by a
↪  unique color, and save it in a configuration file.
```

Figure 3.7: Prompt for mid-session decision 3

The fourth and final mid-session decision was implemented in session 4 after the main content prompt for that session. In all the previous decisions, the agent is asked to modify all existing code to follow that decision (if relevant) due to it otherwise being difficult to evaluate in the end if it is unclear what code has been generated before or after the decision. For this decision however, the agent is asked not to modify any existing code to follow the decision. The reason for this is to force the agent to remember this decision by memory, and not by looking at

already established structure. The mock project contains three pages in total, two of them in cycle 1, and one in cycle 2. This means that the only page that should be effected by this change is the one added in cycle 2. The prompt added can be seen in figure 3.8.

```
I want all new pages pages to be placed inside a
↪  `pages/`-folder for better structure. Don't move existing
↪  pages.
```

Figure 3.8: Prompt for mid-session decision 4

3.3 Context strategies

To be able to improve continuity, a number of different strategies that focused on improving the agents ability to remember previous decisions and/or constraints were developed. These strategies handle the problem in different ways, and is expected to manage different tasks with varying degrees of success. The strategies were determined by the thesis authors in collaboration with the AI team at Bosch R&D Center Lund after multiple brainstorming sessions. The strategies are the following:

1. **To-do list:** With every task the agent receives, divide it into smaller steps, similar to a to-do list, and save the steps in the context-bank with an indicator marking if they were completed or not. In that way, the agent has saved its plan for how to carry out its task. This can be used in later sessions if the agent is to further develop an already existing feature and can then recall its process and decisions the first time.

2. **Decision log:** When the agent notices that a decision has been made during a session, this decision is saved in the context bank for future use. By always reviewing the previous decisions when the agent is to begin new tasks, it can be constantly reminded of the decisions that has been made and can use these in its new work.

3. **Session summary:** At the end of each session, the agent shall summarize the work and conversation that has taken place during the current session. These session summaries are then stored in the context bank for future use. In that way, the agent can remember its work and conversations that have been done before, and can continue its future work along the same lines with similar structure and techniques.

All the context is stored in JSON format in separate files in a context-folder in the mock project (context bank).

3.3.1 Combinations

When testing these 3 strategies, there are 8 different combinations that can be evaluated. To fit the scope of this thesis, only these 5 combinations will be tested:

1. No strategies
2. Only (1) To-do list
3. Only (2) Decision log
4. Only (3) Session summary
5. All three strategies (1, 2, 3)

This selection of combinations were made in collaboration with the thesis authors and the AI team at Bosch R&D Center Lund. Doing it this way allows for a baseline (combination 1) to base the other tests on. There are also tests on the strategies by themselves and a test where all strategies are combined.

3.3.2 Implementation

When evaluating the five different combinations of strategies, the tests were done in three iterations per combination in order to minimize the impact that the stochastic nature of LLM:s can have on the evaluation. After the results were collected, an average result for each combination was calculated.

In order to implement these strategies, a specialized file called `AGENTS.md` was used to inject specific instructions into the Codex CLI agent. This was made by having a base template `AGENTS.md` file (as seen in figure 3.9) and replacing `<rules>` with the injected context management strategies. The structure and content of the template was created iteratively during the process of testing different prompting techniques, as some rules were needed to make sure that generation went smoothly without errors. The actual injection prompts were made by applying relevant prompt engineering strategies [6] such as:

- One-shot
- Contextual prompting
- Chain of thought
- Automatic prompt engineering

Once the base template was finished, as well as the specific strategy injection prompts, a couple of tests were made to ensure that Codex could understand the instructions and apply them correctly. The template and strategy injection prompts were consequently modified based on the test results. As specified earlier, in order to actually implement the strategies, the `<rules>` part of `AGENTS.md` would be replaced with specific injection prompts for every combination.

```
# LLM Dashboard
You are creating a web app containing a dashboard for exploring
↪  LLM usage
```

```
## Compatibility
- Use node.js v18.19.1
- Use Vite v5.4.11
- use Tailwind v3.4.13

## Structure
- requirements.md: Markdown file that contains the requirements
↪  and scope for the project
- agent-context/: Folder where all context files shall be
↪  stored

## Rules
- For every feature and implementation, check requirements.md
↪  to view requirements for that feature and its environment
- When installing any node modules, always make sure that the
↪  version is compatible with your environment and versions
- Never commit anything to git
- Never start the dev server, but always verify that the build
↪  works, especially when adding new libraries

## Context management rules
<rules>
```

Figure 3.9: AGENTS.md base template

For the first combination (No strategies), the agents should not save any additional context. Therefore the text in figure 3.10 will be used as the injection prompt.

```
Don't save any additional context.
```

Figure 3.10: Rules for the agents context management with no strategies

When it comes to the combinations that use specific strategies, every strategy has its own injection prompt that gets used. For strategy 1: To-do list, the agent is instructed to perform the following steps when it is given a new task:

1. Search through existing to-do lists and analyze it for existing patterns and decisions that could impact the current task
2. Create a structured breakdown of the given task, divided into steps to follow
3. Structure the steps into `JSON` format and save it into the context bank

The full injection prompt can be seen in figure 3.11.

```
### Todo list
Before executing any code or modifications, you must initialize
↪  the task context to ensure decisions are documented and
↪  previous logic is reused to maintain continuity.

#### Phase 1: Context Retrieval
1. Search the `agent-context/todo/` directory for existing JSON
↪  context files.
2. Identify patterns, previous architectural decisions, or
↪  completed steps that impact the current task.
3. Explicitly acknowledge any relevant previous context before
↪  proceeding.

#### Phase 2: Planning & Documentation
When a new task is assigned, you shall create a structured
↪  breakdown before writing any functional code:
1. Generate a new JSON file at
↪  `agent-context/todo/{task_name}.json`.
2. Use the following structure for the JSON:
   - `task`: A short text describing the task and its purpose
   - `steps`: A list of objects containing `step`, `status`
   ↪  (pending/done) and `notes` (short comment on what was
   ↪  done and if applicable, how it will impact future tasks)

#### Phase 3: Execution & Sync
1. Update the `status` and `notes` within the JSON file
↪  immediately as you progress through or complete a step.
2. Ensure the JSON file reflects the "ground truth" of the
↪  task's current state at all times.
```

Figure 3.11: Rules for the agents context management with to-do list

For strategy 2: Decision log, the agent shall save a decision log with important decisions. Before the agent starts a new task, it shall look through the previous decisions to see if it has saved any relevant data that may impact the current task. During normal usage, the agent is instructed to watch for decisions made during a session, when the agent notices a decision, it saves it in a decision log. The full injection prompt can be seen in figure 3.12.

```
### Decision logging
```

```
Before executing any code or modifications, you must consult
↪  the persistent decision log to ensure alignment with
↪  established patterns. Every significant technical choice
↪  made during this session must be recorded for future
↪  context to maintain continuity.

#### Phase 1: Context retrieval
1. Scan `agent-context/decisions/` for all existing JSON
↪  context files
2. Identify any specific architectural constraints, patterns,
↪  or logical decisions that apply to the current request

#### Phase 2: Decision capture
Whenever a decision, rule or design choice is made by the user,
↪  document it immediately
1. Create or update a JSON file at
↪  `agent-context/decisions/{feature_area}.json`
2. Use the following structure for the JSON:
   - `decision`: Short name for the decision
   - `description`: Describe the decision in required level of
   ↪  detail

#### Phase 3: Session summary
At the end of a task, verify that the
↪  `agent-context/decisions/` folder accurately reflects the
↪  current state of the application's architecture
```

Figure 3.12: Rules for the agents context management with decision log

For strategy 3: Session summary, the agent shall summarize the session at the end of a session. When the agent is given instructions to begin a new task it needs to read through previous session summaries to see if anything is applicable for this task. Due to limitations in Codex CLI, the user needs to write the phrase "EXIT SESSION" in order to let the agent know that the session is ending. The full injection prompt can be seen in figure 3.13.

```
### Session summarization
Before beginning any work you must review the history of
↪  previous sessions to maintain continuity. At the conclusion
↪  of the current session (when the user writes `EXIT
↪  SESSION`), you must write down all actions into a concise
↪  summary for future use.

#### Phase 1: Context reconstruction
```

```
1. Search the `agent-context/sessions/` directory for existing
↪  JSON summary files.
2. Review the `summary` and `pending_threads` from the previous
↪  session to understand the immediate starting point.

#### Phase 2: Activity tracking
Throughout the session, maintain an internal log of:
1. Significant file changes or new components created.
2. Problems encountered and how they were addressed.
3. Specific user preferences or instructions shared during the
↪  dialogue.

#### Phase 3: Handoff generation
When the user types `EXIT SESSION`, you must create a JSON file
↪  at `agent-context/sessions/session_{timestamp}.json` using
↪  the following structure:
- `summary`: A high-level overview of what was achieved during
↪  this session
- `changes`: A list of files modified
- `pending_threads`: Specific tasks or unresolved logic for
↪  future sessions to address.
```

Figure 3.13: Rules for the agents context management with session summary

When evaluating a combination of different strategies, all the commands for the strategies to test were added after each other instead of <rules> in the template. For a view of how the complete `AGENTS.md` files look like with the injection prompts, see the following figures in appendix D:

1. **No strategies:** Figure D.1.
2. **Only (1) To-do list:** Figure D.2
3. **Only (2) Decision log:** Figure D.3
4. **Only (3) Session summary:** Figure D.4
5. **All three strategies (1, 2, 3):** Figure D.5

3.4 Data collection and evaluation

The data that was collected for this thesis was as follows:

1. The generated code for the web app.
2. The time it takes to generate the web app.
3. The number of tokens the agent is using to generate the web app. This is then converted into an actual cost in euro.

To evaluate and compare the continuity of the different combination of strategies, the following four metrics are assessed:

- **Requirement compliance:** This metric measures how well the web app meets the requirement specification.
- **Decision consistency:** This assesses how accurately the agent remembers mid-session decisions.
- **Generation cost:** To evaluate resource usage and cost-effectiveness, the tokens used for generation is converted into an actual cost.
- **Temporal efficiency:** This metric is the total generation time in seconds.

3.4.1 Method of evaluation

The different metrics were evaluated in different ways, depending on the metric. Some metrics were evaluated manually by the thesis authors by observing generated code and output and matching it to specified criteria.

When calculating an average result in this thesis, formula 3.1 is used.

$$A = \frac{1}{n} \sum_{i=1}^{n} a_i \tag{3.1}$$

In formula 3.1, A is the calculated average, n is the number of values and a_i is the data set values.

Requirement compliance

Evaluation of requirement compliance was carried out via manual testing, where the proportion of implemented requirements in relation to the total amount of requirements was calculated. This was done by going through the requirement specification after cycle 2 was generated, and observing how many requirements were followed in the generated web app. For example, there are 37 requirements in total. If combination 1, iteration 2 only implemented 10 requirements correctly, that iterations requirement compliance is 27.03 %. The average compliance percentage for a combination was then calculated by taking the average of every iterations compliance percentage with formula in 3.1. All requirements that were not followed in every iteration were also documented to see if any requirement was harder for the agent to achieve.

Decision consistency

For evaluating decision consistency, the thesis authors manually looked though the code and visually observed the generated web app for every iteration. This was then used to calculate a degree of success by taking the percentage of instances where the decision was correctly applied versus the total number of applicable scenarios. To evaluate the long-term adherence to a decision, only sessions occurring after its initial introduction were analyzed. This approach ensures that the evaluation measures the agents ability to adhere to the decisions over time, rather than

its immediate response to the prompt. The average decision consistency for each strategy combination was then calculated by taking the average iterations decision consistency percentage using formula 3.1. This process was repeated separately for every decision. Both cycle 1 and 2 were recorded in this way for decisions 1, 2, and 3, while decision 4 was only recorded for cycle 2. This was due to decision 4 not being applicable in cycle 1.

Generation costs

When generating the mock project, token consumption is divided into completion tokens and prompt tokens. These tokens each have a specified real-life cost assigned to them that is costing Bosch on every use. In order to calculate costs, the number of completion and prompt tokens were collected every time a prompt was inputted and processed. Each iterations prompt tokens and completion tokens were then summarized over both cycles. The total amount of tokens for each iteration per type was then used in formula 3.1 to calculate the average number of tokens for each token type. The average number of tokens per type was then entered into formula 3.2 to calculate the total generation cost for each combination.

$$x = \frac{y}{10^6} \cdot 0.22 + \frac{z}{10^6} \cdot 1.74 \tag{3.2}$$

In formula 3.2, x is the total generation cost in euro, y is the amount of input tokens (also described as prompt tokens in this thesis), and z is the amount of output tokens (also described as completion tokens in this thesis). Due to the fluctuating costs of tokens, these costs were taken from a specific snapshot in time, specifically 2026-04-22 15:06:20 [25]. The version of codex used in this thesis was GPT-5-mini Global so the pricing collected was for that LLM. This timestamp was decided by the AI team at Bosch R&D Center Lund. The pricing for 1 million tokens at this timestamp was 0.22 EUR for input tokens and 1.74 EUR for output tokens.

Temporal efficiency

For the temporal efficiency, the total generation time for each iteration was calculated by summarizing the total generation time for every prompt over both cycles. An average generation time for each combination was then calculated by using formula 3.1 with the generation time for each iteration.

Generation errors

When generating code during a session, the agent sometimes makes errors that result in the web app not working as intended. During the cases where these errors affect the web app in such a way that the app can not be started or pages refuse to load, that specific session is re-run once. This is only done once in order to counter the stochastic nature of generation, meaning that the error can be seen as a fluke. However, if the error persists even after re-generation, it can be seen as a pattern resulting from the specific context management strategies being implemented at the time. In those cases, the agent is asked to fix the error, with the fix-time and

tokens being recorded and appended to that sessions generation time and token usage, as well as the error being noted during data collection.

This method of error handling is however only applied when the errors are deeply severe (as described above), and is not used when the agent is simply not following instructions and/or requirements.

3.5 Method of analysis

When analyzing the results, the most important metrics to assess according to continuity are requirement compliance and decision consistency. Generation cost is not related to continuity, but is an important viewpoint for Bosch due to their interest in lowering expenses. Temporal efficiency is, like generation cost, not related to continuity but is an important qualification metric that can be used to evaluate two different combinations with similar results on the other metrics.

3.6 Communication

This chapter describes the communication between the authors, with the company and with the supervisor.

3.6.1 Communication between authors

While working on the thesis the authors were seated next to each other at Bosch R&D Center Lund. The authors had both formal and informal meetings with each other to discuss ideas and future work. Most of the work with prompting and coding were done using pair programming, where both authors developed ideas together.

3.6.2 Communication with the company

Communication with the company was done by the authors sittning in the office near the mentors. This allowed for easy verbal communication when needed. For other forms of communication, Microsoft Teams was used as a communication channel for meetings and chatting. There were also weekly meetings between the authors and the company mentors for regular checkups to make sure everything was on track.

3.6.3 Communication with supervisor

Communication with the supervisor was done primarily by mail and meetings every other week. This allowed regular checkups and getting feedback of the thesis.

3.7 Source critique

During this thesis, many different kinds of sources has been used. In order to evaluate credibility for all of the sources in this report, this chapter presents all

sources divided into subgroups where each subgroup is evaluated separately.

3.7.1 Bosch AI Team

Sources: [1]

This source is used once in the report, where the thesis describes how Bosch is currently using AI agents in development. This is therefore a credible source due to it being their own product.

3.7.2 Online sources

Sources: [2], [3], [4], [5], [7], [8], [9], [10], [11], [12], [24], [26]

This group of sources consists of various companies describing key concepts about how their own AI products and/or services work, companies such as IBM, NASA, Microsoft, and Engineering at Anthropic. Normally this would be something to look out for, as companies usually wants to present their own products in a favorable way, and may therefore not mention undesirable factors. However, for this thesis, these sources were only used in order have a reference when describing key concepts and well known general terms that are used in AI development. This means that these sources were only used to describe something factual, and not anything opinionated.

3.7.3 Reports

Sources: [6], [27], [28], [29], [30], [31]

This group of sources consists of various scientific reports, and are used critically in this thesis in order to fact-check and confirm findings and methods used. It is therefore important that these sources are factual and correct.

Sources [27], [28] are both peer reviewed academic literature, published in highly regarded journals such as PNAS and IEEE/ACM Alware conference, this ensures a high level of credibility. These sources are also very recent (2024 and 2025), which is necessary to stay relevant in the AI field that is currently very fast-moving.

The report [6] is a technical guide. While it comes from a company, it is used here only to reference industry-standard methods for prompt engineering.

The sources [29], [30], [31], have not undergone the same rigorous peer-review process. These should be viewed with more caution as they may lack external validation. However, they are used in this thesis to provide a current perspective on the rapidly evolving AI field and its environmental impact. This is only a small part of the thesis mentioning ethical viewpoints, and is not foundational to the point the thesis is making.

3.7.4 Product pages

Sources: [13], [14], [15], [16], [17], [18], [19], [20], [21], [22], [23], [25]

This group of sources consists of product pages, usually the *about* page. These product descriptions all come from the company that owns the product, meaning that it is open to impartial descriptions. However, their use in this thesis is

limited to being a reference the readers can go to if they want to learn more about a product that was used in this thesis. They are not used for fact checking or to form any kind of opinionated result or method, they are simply descriptions of products and services used in this thesis.

3.8 Use of AI

During this thesis, different AI tools were used. ChatGPT and Gemini were used to help find relevant references, all references were manually read by the authors to assess their relevancy. ChatGPT and Gemini were also used as a sparring partner to bounce ideas off of for structure. NotebookLM with Gemini was used to clarify prompt engineering structure and optimal usage. Writefull was used to correct grammar and spelling. The thesis authors ensure that all work in this thesis was done by them and that AI was only used as a support tool.

Chapter 4
Results

This chapter describes the results gathered from the generated mock project. Each combination of context strategies has been generated in three iterations, the average percentages for requirement compliance and decision consistency is presented in tables under each subsection. The average generative cost and temporal efficiency is also presented in tables under each subsection.

For more detailed description of how the result is gathered and what each part means, see chapter 3.4.1.

4.1 Combination 1

This chapter shows the results for combination 1: No strategies. This combination used no context management strategies, and is used as a baseline to compare the other strategies against.

4.1.1 Requirement compliance - Combination 1

The requirement compliance results for combination 1 can be seen in the table 4.1.

For this combination, the following requirements were missing in the following number of iterations:

- **One iteration:** R-UI-02, R-UI-05, R-HMP-06, R-EV-02 and R-CO-04.
- **Two iterations:** R-EV-01.
- **All three iterations:** No requirements were missing in all three iterations.

For detailed descriptions of each requirement, see appendix A, figure A.2.

4.1.2 Decision consistency - Combination 1

For more details on every decision, see chapter 3.2.5, and for information on how the results are collected and evaluated, see chapter 3.4.1

The decision consistency for decision 1 can be seen in table 4.2. Decision 1 asked the agent to append the text "(Heading)" to the end of every heading.

The decision consistency for decision 2 can be seen in table 4.3. Decision 2 asked the agent to add a comment containing the file creation date at the top of each `.tsx` and `.ts` file.

Table 4.1: Combination 1: Degree of success for requirement compliance

	Degree of success
Iteration 1	97.30 %
Iteration 2	91.89 %
Iteration 3	91.89 %
Average	93.69 %

Table 4.2: Combination 1: Degree of success for complying with Decision 1: Add the text "(Heading)" to each heading

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	0.00 %	0.00 %
Iteration 3	0.00 %	0.00 %
Average	0.00 %	0.00 %

Table 4.3: Combination 1: Degree of success for complying with Decision 2: Add a comment at the top of each .tsx and .ts file

	Cycle 1	**Cycle 2**
Iteration 1	33.33 %	0.00 %
Iteration 2	14.29 %	0.00 %
Iteration 3	25.00 %	0.00 %
Average	24.21 %	0.00 %

The decision consistency for decision 3 can be seen in table 4.4. Decision 3 asked the agent to create a color palette for every model used, and to use that color at every model mention. A config file for the model colors exists in every iteration but is not used in any of the iterations.

Table 4.4: Combination 1: Degree of success for complying with Decision 3: Create a color palette and implement it at every model mention

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	0.00 %	0.00 %
Iteration 3	0.00 %	0.00 %
Average	0.00 %	0.00 %

The decision consistency for decision 4 can be seen in table 4.5. Decision 4 asked the agent to place the comparison page in a pages folder. In all iterations,

all pages were placed in a pages folder, not only the comparison page.

Table 4.5: Combination 1: Degree of success for complying with Decision 4: Implement comparison page in a pages folder

	Degree of success
Iteration 1	100.00 %
Iteration 2	100.00 %
Iteration 3	100.00 %
Average	100.00 %

4.1.3 Generation costs - Combination 1

The average amount of prompt tokens for combination 1 can be seen in table 4.6. The average amount of completion tokens for combination 1 can be seen in table 4.7. By inputting the results into formula 3.2, the total average generation cost for this combination can be calculated to 0.60 EUR.

Table 4.6: Combination 1: Prompt token consumption

	Prompt tokens
Iteration 1	2 799 470
Iteration 2	2 734 412
Iteration 3	2 498 800
Average	2 677 560.67

Table 4.7: Combination 1: Completion token consumption

	Completion tokens
Iteration 1	7 377
Iteration 2	7 355
Iteration 3	6 934
Average	7 222

4.1.4 Temporal efficiency - Combination 1

The results of total generation time for combination 1 can be seen in table 4.8. The average generation time for this combination resulted in 1 887 seconds.

4.1.5 Generation errors - Combination 1

In this combination there were no errors during generation of the mock project in any of the iterations.

Table 4.8: Combination 1: Temporal efficiency

	Temporal efficiency (sec)
Iteration 1	1 659
Iteration 2	2 006
Iteration 3	1 996
Average	1 887

4.2 Combination 2

This chapter shows the results for combination 2: (1) To-do list. This combination uses the strategy To-do list, a strategy that lets the agent break every task down into several subtasks. For more information on this strategy, see chapter 3.3.

4.2.1 Requirement compliance - Combination 2

The requirement compliance results for combination 2 can be seen in the table 4.9.

For this combination, the following requirements were missing in the following number of iterations:

- **One iteration:** R-UI-02, R-FLT-01
- **Two iterations:** R-UI-03, R-HMP-04, R-HMP-06
- **All three iterations:** R-UI-05, R-EV-01, R-CO-04

For detailed descriptions of each requirement, see appendix A, figure A.2.

Table 4.9: Combination 2: Degree of success for requirement compliance

	Degree of success
Iteration 1	83.78 %
Iteration 2	86.49 %
Iteration 3	86.49 %
Average	85.59 %

4.2.2 Decision consistency - Combination 2

For more details on every decision, see chapter 3.2.5, and for information on how the results are collected and evaluated, see chapter 3.4.1

The decision consistency for decision 1 can be seen in table 4.10. Decision 1 asked the agent to append the text "(Heading)" to the end of every heading.

The decision consistency for decision 2 can be seen in table 4.11. Decision 2 asked the agent to add a comment containing the file creation date at the top of each `.tsx` and `.ts` file.

Table 4.10: Combination 2: Degree of success for complying with Decision 1: Add the text "(Heading)" to each heading

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	0.00 %	0.00 %
Iteration 3	0.00 %	0.00 %
Average	0.00 %	0.00 %

Table 4.11: Combination 2: Degree of success for complying with Decision 2: Add a comment at the top of each .tsx and .ts file

	Cycle 1	**Cycle 2**
Iteration 1	14.29 %	33.33 %
Iteration 2	66.67 %	50.00 %
Iteration 3	33.33 %	0.00 %
Average	38.10 %	27.78 %

The decision consistency for decision 3 can be seen in table 4.12. Decision 3 asked the agent to create a color palette for every model used, and to use that color at every model mention. A config file for the model colors exists in every iteration, but its application varies.

Table 4.12: Combination 2: Degree of success for complying with Decision 3: Create a color palette and implement it at every model mention

	Cycle 1	**Cycle 2**
Iteration 1	50.00 %	100.00 %
Iteration 2	0.00 %	0.00 %
Iteration 3	50.00 %	100.00 %
Average	33.33 %	66.67 %

The decision consistency for decision 4 can be seen in table 4.13. Decision 4 asked the agent to place the comparison page in a pages folder. In all iterations, all pages were placed in a pages folder, not only the comparison page.

4.2.3 Generation costs - Combination 2

The average amount of prompt tokens for combination 2 can be seen in table 4.14. The average amount of completion tokens for combination 2 can be seen in table 4.15. By inputting the results into formula 3.2, the total average generation cost for this combination can be calculated to 0.76 EUR.

Table 4.13: Combination 2: Degree of success for complying with Decision 4: Implement comparison page in a pages folder

	Degree of success
Iteration 1	100.00 %
Iteration 2	100.00 %
Iteration 3	100.00 %
Average	100.00 %

Table 4.14: Combination 2: Prompt token consumption

	Prompt tokens
Iteration 1	3 501 130
Iteration 2	3 194 100
Iteration 3	3 403 570
Average	3 366 266.67

Table 4.15: Combination 2: Completion token consumption

	Completion tokens
Iteration 1	8 480
Iteration 2	8 943
Iteration 3	8 434
Average	8 619

4.2.4 Temporal efficiency - Combination 2

The results of total generation time for combination 2 can be seen in table 4.16. The average generation time for this combination resulted in 2 172 seconds.

Table 4.16: Combination 2: Temporal efficiency

	Temporal efficiency (sec)
Iteration 1	2 129
Iteration 2	2 186
Iteration 3	2 201
Average	2 172

4.2.5 Generation errors - Combination 2

In this combination there were no errors during generation of the mock project in any of the iterations.

4.3 Combination 3

This chapter shows the results for combination 3: (2) Decision log. This combination uses the strategy Decision log, which lets the agent store all decisions that are made during a session. For more information on this strategy, see chapter 3.3.

4.3.1 Requirement compliance - Combination 3

The requirement compliance results for combination 3 can be seen in the table 4.17.

For this combination, the following requirements were missing in the following number of iterations:

- **One iteration:** R-HMP-04, R-HMP-06, R-EV-01, R-CO-03, R-EX-03, R-FLT-01 and R-FLT-03.
- **Two iterations:** R-UI-02, R-UI-05 and R-FLT-02.
- **All three iterations:** No requirements were missing in all three iterations.

For detailed descriptions of each requirement, see appendix A, figure A.2.

Table 4.17: Combination 3: Degree of success for requirement compliance

	Degree of success
Iteration 1	89.19 %
Iteration 2	81.08 %
Iteration 3	94.59 %
Average	88.29 %

4.3.2 Decision consistency - Combination 3

For more details on every decision, see chapter 3.2.5, and for information on how the results are collected and evaluated, see chapter 3.4.1

The decision consistency for decision 1 can be seen in table 4.18. Decision 1 asked the agent to append the text "(Heading)" to the end of every heading.

Table 4.18: Combination 3: Degree of success for complying with Decision 1: Add the text "(Heading)" to each heading

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	0.00 %	100.00 %
Iteration 3	0.00 %	0.00 %
Average	0.00 %	33.33 %

The decision consistency for decision 2 can be seen in table 4.19. Decision 2 asked the agent to add a comment containing the file creation date at the top of each `.tsx` and `.ts` file.

Table 4.19: Combination 3: Degree of success for complying with Decision 2: Add a comment at the top of each .tsx and .ts file

	Cycle 1	**Cycle 2**
Iteration 1	33.33 %	50.00 %
Iteration 2	44.44 %	33.33 %
Iteration 3	80.00 %	100.00 %
Average	52.59 %	61.11 %

The decision consistency for decision 3 can be seen in table 4.20. Decision 3 asked the agent to create a color palette for every model used, and to use that color at every model mention. A config file for the model colors exists in every iteration, but its application varies.

Table 4.20: Combination 3: Degree of success for complying with Decision 3: Create a color palette and implement it at every model mention

	Cycle 1	**Cycle 2**
Iteration 1	50.00 %	100.00 %
Iteration 2	50.00 %	100.00 %
Iteration 3	50.00 %	100.00 %
Average	50.00 %	100.00 %

The decision consistency for decision 4 can be seen in table 4.21. Decision 4 asked the agent to place the comparison page in a pages folder. In all iterations, all pages were placed in a pages folder, not only the comparison page.

Table 4.21: Combination 3: Degree of success for complying with Decision 4: Implement comparison page in a pages folder

	Degree of success
Iteration 1	100.00 %
Iteration 2	100.00 %
Iteration 3	100.00 %
Average	100.00 %

4.3.3 Generation costs - Combination 3

The average amount of prompt tokens for combination 3 can be seen in table 4.22. The average amount of completion tokens for combination 3 can be seen in table

4.23. By inputting the results into formula 3.2, the total average generation cost for this combination can be calculated to 0.71 EUR.

Table 4.22: Combination 3: Prompt token consumption

	Prompt tokens
Iteration 1	2 797 430
Iteration 2	3 105 230
Iteration 3	3 567 860
Average	3 156 840

Table 4.23: Combination 3: Completion token consumption

	Completion tokens
Iteration 1	7 601
Iteration 2	8 210
Iteration 3	9 312
Average	8 374.33

4.3.4 Temporal efficiency - Combination 3

The results of total generation time for combination 3 can be seen in table 4.24. The average generation time for this combination resulted in 2 090 seconds.

Table 4.24: Combination 3: Temporal efficiency

	Temporal efficiency (sec)
Iteration 1	1 689
Iteration 2	2 198
Iteration 3	2 383
Average	2 090

4.3.5 Generation errors - Combination 3

In this combination there were no errors during generation of the mock project in any of the iterations.

4.4 Combination 4

This chapter shows the results for combination 4: (3) Session summary. This combination uses the strategy Session summary, which lets the agent store a summary of an entire session at the end of the session. For more information on this strategy, see chapter 3.3.

4.4.1 Requirement compliance - Combination 4

The requirement compliance results for combination 4 can be seen in the table 4.25.

For this combination, the following requirements were missing in the following number of iterations:

- **One iteration:** R-UI-02, R-UI-05, R-HMP-04, R-HMP-06, R-EV-04 and R-CO-04.
- **Two iterations:** No requirements were missing in two of the iterations.
- **All three iterations:** No requirements were missing in all three iterations.

For detailed descriptions of each requirement, see appendix A, figure A.2.

Table 4.25: Combination 4: Degree of success for requirement compliance

	Degree of success
Iteration 1	94.59 %
Iteration 2	97.30 %
Iteration 3	91.89 %
Average	94.59 %

4.4.2 Decision consistency - Combination 4

For more details on every decision, see chapter 3.2.5, and for information on how the results are collected and evaluated, see chapter 3.4.1

The decision consistency for decision 1 can be seen in table 4.26. Decision 1 asked the agent to append the text "(Heading)" to the end of every heading.

Table 4.26: Combination 4: Degree of success for complying with Decision 1: Add the text "(Heading)" to each heading

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	0.00 %	0.00 %
Iteration 3	0.00 %	100.00 %
Average	0.00 %	33.33 %

The decision consistency for decision 2 can be seen in table 4.27. Decision 2 asked the agent to add a comment containing the file creation date at the top of each `.tsx` and `.ts` file.

The decision consistency for decision 3 can be seen in table 4.28. Decision 3 asked the agent to create a color palette for every model used, and to use that color at every model mention. A config file for the model colors exists in every iteration, but its application varies.

Table 4.27: Combination 4: Degree of success for complying with Decision 2: Add a comment at the top of each .tsx and .ts file

	Cycle 1	**Cycle 2**
Iteration 1	66.67 %	33.33 %
Iteration 2	37.50 %	0.00 %
Iteration 3	0.00 %	0.00 %
Average	34.72 %	11.11 %

Table 4.28: Combination 4: Degree of success for complying with Decision 3: Create a color palette and implement it at every model mention

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	50.00 %	100.00 %
Iteration 3	0.00 %	0.00 %
Average	16.67 %	33.33 %

The decision consistency for decision 4 can be seen in table 4.29. Decision 4 asked the agent to place the comparison page in a pages folder. In all iterations, all pages were placed in a pages folder, not only the comparison page.

Table 4.29: Combination 4: Degree of success for complying with Decision 4: Implement comparison page in a pages folder

	Degree of success
Iteration 1	100.00 %
Iteration 2	100.00 %
Iteration 3	100.00 %
Average	100.00 %

4.4.3 Generation costs - Combination 4

The average amount of prompt tokens for combination 4 can be seen in table 4.30. The average amount of completion tokens for combination 4 can be seen in table 4.31. By inputting the results into formula 3.2, the total average generation cost for this combination can be calculated to 0.84 EUR.

4.4.4 Temporal efficiency - Combination 4

The results of total generation time for combination 4 can be seen in table 4.32. The average generation time for this combination resulted in 2 758.33 seconds.

Table 4.30: Combination 4: Prompt token consumption

	Prompt tokens
Iteration 1	3 788 910
Iteration 2	3 749 150
Iteration 3	3 762 300
Average	3 766 786.67

Table 4.31: Combination 4: Completion token consumption

	Completion tokens
Iteration 1	9 292
Iteration 2	8 313
Iteration 3	9 162
Average	8 922.33

Table 4.32: Combination 4: Temporal efficiency

	Temporal efficiency (sec)
Iteration 1	2 349
Iteration 2	2 807
Iteration 3	3 119
Average	2 758.33

4.4.5 Generation errors - Combination 4

In this combination the following errors were discovered:

- **Single error leading to regeneration:** No error leading to a fix-prompt.
- **Multiple errors leading to regeneration and fix-prompt:** One time in one iteration.

4.5 Combination 5

This chapter shows the results for combination 5: All three strategies. This combination uses all context management strategies at the same. For more information on how these strategies work, see chapter 3.3.

4.5.1 Requirement compliance - Combination 5

The requirement compliance results for combination 5 can be seen in the table 4.33.

For this combination, the following requirements were missing in the following number of iterations:

- **One iteration:** R-UI-02, R-UI-03, R-HMP-01, R-HMP-04, R-HMP-05, R-EV-02, R-EX-03, R-FLT-01 and R-FLT-03.
- **Two iterations:** R-HMP-06.
- **All three iterations:** R-UI-05 and R-EV-01.

For detailed descriptions of each requirement, see appendix A, figure A.2.

Table 4.33: Combination 5: Degree of success for requirement compliance

	Degree of success
Iteration 1	81.08 %
Iteration 2	89.19 %
Iteration 3	83.78 %
Average	84.68 %

4.5.2 Decision consistency - Combination 5

For more details on every decision, see chapter 3.2.5, and for information on how the results are collected and evaluated, see chapter 3.4.1

The decision consistency for decision 1 can be seen in table 4.34. Decision 1 asked the agent to append the text "(Heading)" to the end of every heading.

Table 4.34: Combination 5: Degree of success for complying with Decision 1: Add the text "(Heading)" to each heading

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	0.00 %	0.00 %
Iteration 3	0.00 %	0.00 %
Average	0.00 %	0.00 %

The decision consistency for decision 2 can be seen in table 4.35. Decision 2 asked the agent to add a comment containing the file creation date at the top of each `.tsx` and `.ts` file.

Table 4.35: Combination 5: Degree of success for complying with Decision 2: Add a comment at the top of each .tsx and .ts file

	Cycle 1	**Cycle 2**
Iteration 1	25.00 %	0.00 %
Iteration 2	20.00 %	33.33 %
Iteration 3	0.00 %	50.00 %
Average	15.00 %	27.78 %

The decision consistency for decision 3 can be seen in table 4.36. Decision 3 asked the agent to create a color palette for every model used, and to use that color at every model mention. A config file for the model colors exists in every iteration, but its application varies.

Table 4.36: Combination 5: Degree of success for complying with Decision 3: Create a color palette and implement it at every model mention

	Cycle 1	**Cycle 2**
Iteration 1	0.00 %	0.00 %
Iteration 2	50.00 %	100.00 %
Iteration 3	50.00 %	100.00 %
Average	33.33 %	66.67 %

The decision consistency for decision 4 can be seen in table 4.37. Decision 4 asked the agent to place the comparison page in a pages folder. In all iterations, all pages were placed in a pages folder, not only the comparison page.

Table 4.37: Combination 5: Degree of success for complying with Decision 4: Implement comparison page in a pages folder

	Degree of success
Iteration 1	100.00 %
Iteration 2	100.00 %
Iteration 3	100.00 %
Average	100.00 %

4.5.3 Generation costs - Combination 5

The average amount of prompt tokens for combination 5 can be seen in table 4.38. The average amount of completion tokens for combination 5 can be seen in table 4.39. By inputting the results into formula 3.2, the total average generation cost for this combination can be calculated to 0.91 EUR.

Table 4.38: Combination 5: Prompt token consumption

	Prompt tokens
Iteration 1	4 383 790
Iteration 2	4 054 570
Iteration 3	3 792 320
Average	4 076 893.33

Table 4.39: Combination 5: Completion token consumption

	Completion tokens
Iteration 1	10 189
Iteration 2	10 537
Iteration 3	9 806
Average	10 177.33

4.5.4 Temporal efficiency - Combination 5

The results of total generation time for combination 5 can be seen in table 4.40. The average generation time for this combination resulted in 3 898.67 seconds.

Table 4.40: Combination 5: Temporal efficiency

	Temporal efficiency (sec)
Iteration 1	4 106
Iteration 2	4 228
Iteration 3	3 362
Average	3 898.67

4.5.5 Generation errors - Combination 5

In this combination the following errors were discovered:

- **Single error leading to regeneration:** Two total times in two iterations.
- **Multiple errors leading to regeneration and fix-prompt:** One time in one iteration.

4.6 Summary

This chapter showcases summaries of the results from all combinations, using charts to display the average values for each combination in requirement compliance, decision consistency, generation cost and temporal efficiency.

4.6.1 Requirement compliance - Summary

The average requirement compliance for all combinations can be seen in figure 4.1. For a display over which requirements were missed and how often in each combination, see figure 4.2.

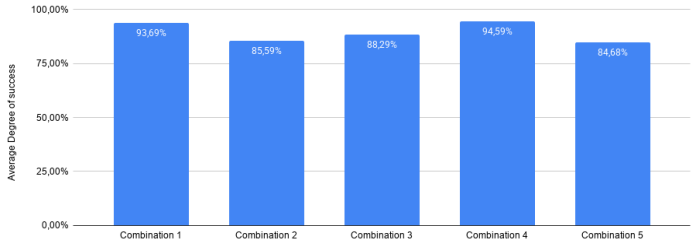

Figure 4.1: Average requirement compliance

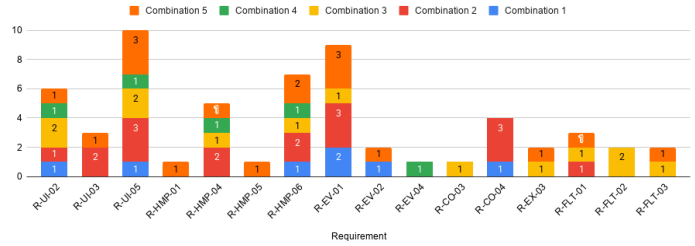

Figure 4.2: Number of times a requirement was missed in each combination

4.6.2 Decision consistency - Summary

The average decision consistency across all combinations for decision 1 in each combination can be seen in figure 4.3. Decision 1 asked the agent to append the text "(Heading)" to the end of every heading.

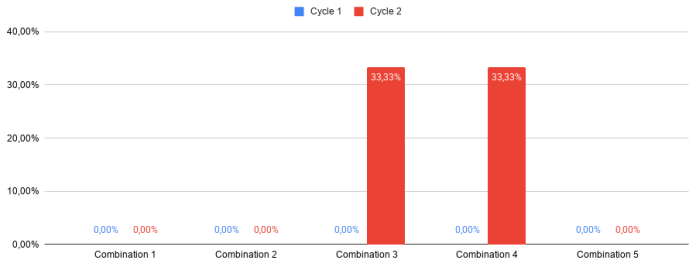

Figure 4.3: Average decision compliance for decision 1

The average decision consistency across all combinations for decision 2 can be seen in figure 4.4. Decision 2 asked the agent to add a comment containing the file creation date at the top of each `.tsx` and `.ts` file.

The average decision consistency across all combinations for decision 3 can be seen in figure 4.5. Decision 3 asked the agent to create a color palette for every model used, and to use that color at every model mention. A config file for the

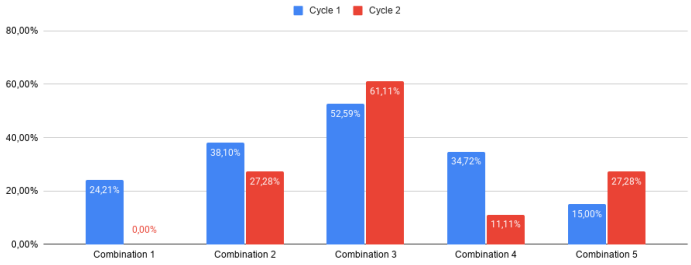

Figure 4.4: Average decision compliance for decision 2

model colors exists in every iteration, but its application varies.

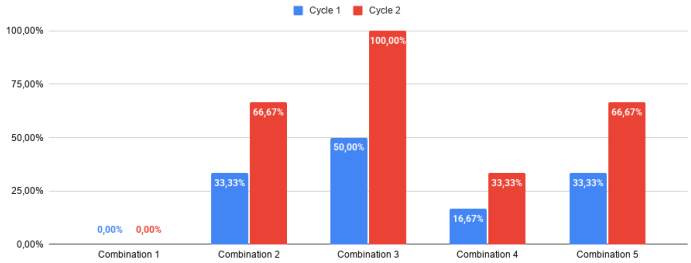

Figure 4.5: Average decision compliance for decision 3

The average decision consistency across all combinations for decision 4 can be seen in figure 4.6. Decision 4 asked the agent to place the comparison page in a pages folder. In all iterations, all pages were placed in a pages folder, not only the comparison page.

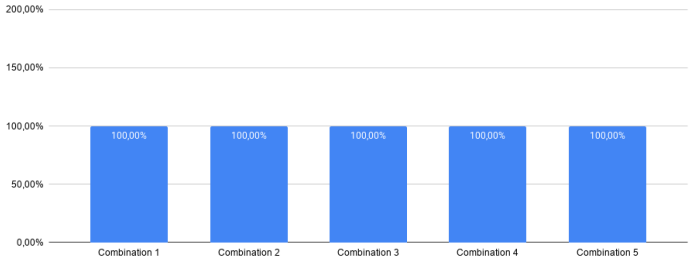

Figure 4.6: Average decision compliance for decision 4

4.6.3 Generation cost - Summary

The average generation costs across all combinations can be seen in figure 4.7.

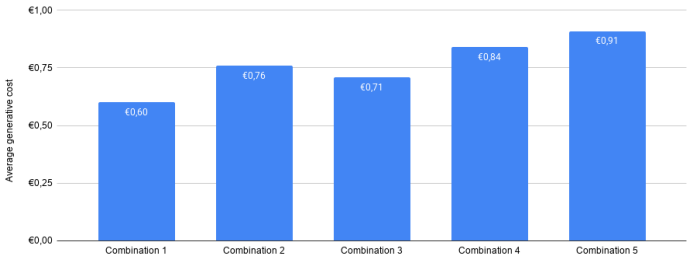

Figure 4.7: Average generation costs

4.6.4 Temporal efficiency - Summary

The average generation time across all combinations be seen in figure 4.8.

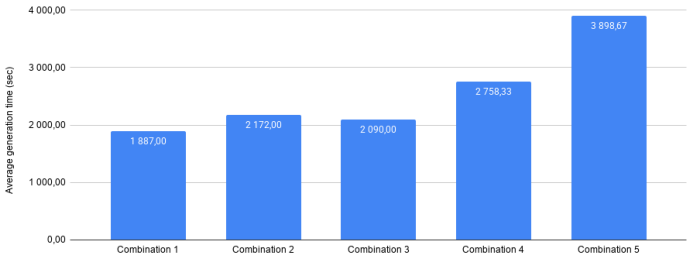

Figure 4.8: Average generation time

4.6.5 Generation errors - Summary

The amount of generation errors that occurred during generation of each combination can be seen in figure 4.9.

Figure 4.9: Summary of errors when generating the mock project

Chapter 5
Discussion and conclusion

This chapter analyzes the results shown in chapter 4 to see what conclusions can be drawn. Furthermore, it provides a critical evaluation of the chosen methodology, addressing potential limitations and suggestions for future development. Finally, the chapter reflects on the ethical aspects relevant to the usage of AI.

5.1 Discussion

During agentic development, there are many factors that can impact the generated results. This could lead to results being misleading and difficult to interpret. This chapter raises some of these issues, and what could be done to mitigate them.

5.1.1 Difficulties in decision adherence

During evaluation of the mid-session results in this thesis, decision 1 (append "(heading)" to every heading) was noted to be difficult for the agent to follow, with a success rate of only 33.33 % in only combination 3 and 4. This is especially apparent compared to decision 2 (add creation timestamp at the top of `.ts` and `.tsx` files), which has a success rate that varies between 11.11 % to 61.11 % in all combinations. The reasoning behind this sharp decline in success could be attributed to training data bias, where the LLM that powers the agent has an easier time applying methods that are found more frequently in its training data [27]. In this case, adding a creation timestamp at the top of a file might be a more common practice in the training data than appending an arbitrary attachment to all headings.

The agent's difficulty with complying with decision 1 compared to decision 2 could also stem from insufficient prompt wording. The specific wording of a prompt has been found to strongly affect an LLM:s generated output, to the point where incorrect or inadequate prompts could lead to the LLM making incorrect assumptions, leading to incorrect results [28]. The wording when introducing decision 1 (as seen in session 1, prompt 2 in appendix C, figure C.1) may not adequately clarify that this decision should also apply to all future headings, causing the agent to only apply the decision to existing headings. This becomes more apparent when comparing with the wording of decision 2 (as seen in session 2, prompt 2 in appendix C, figure C.1) which alludes more clearly to the decision being applied in

future applications.

5.1.2 Time- and token usage variation

As seen in the results, token- and time usage vary slighty, depending on which context generation strategies were being used. However, it should be noted that these metrics may have been affected by more factors than just context management strategies. Factors such as time of day, the agent's current work load from other users, or network latency from accessing the agent's server via VPN, could all play a part in the gathered metrics. In this thesis, most iterations for one combination were generated over the same day, meaning that if the agent's servers were under heavy load from other users that day, the generation times would suffer. In order to mitigate these variables, all testing would have to have been done in an isolated environment, where the external variables would not vary between project iterations. However, this would require much more time and planning than what was possible for this thesis.

5.2 Problem statements

This chapter is a reflection of whether or not the problem statements were answered in this thesis or not.

The focus of this thesis has shifted over time. At the beginning, Bosch wanted to analyze different strategies for the different parts of the session (pre-, intra-, and post-session). However, due to time constraints, this focus was shifted to selecting and analyzing three context strategies in total without focusing on the specific parts of the session. This has caused some of the original problem statements to become difficult to answer in the way that was originally intended.

1. **How is context management currently handled in continuous software development tasks in two already established agentic softwares?**

   This question was found to be difficult to answer right from the start of the thesis work. Methods and techniques for how the different commercially available agentic tools manage context are not generally publicized, and therefore this question could not be answered.

2. **Which pre-session context management strategies are suitable for initializing AI agents?**

   Pre-session strategies were generally excluded from the context management testing. Although combinations 2 - 5 do use a pre-session analysis of the context bank before starting any task, these were not specifically targeted due to the shift in thesis focus. Therefore, no conclusions about pre-session strategies can be made from this thesis.

3. **Which intra-session context management strategies are suitable for dynamic context retrieval?**

   This thesis has tested two intra-session strategies, strategy 1: To-do list and strategy 2: Decision log. This means that combination 2, 3, and 5

all used some forms of intra-session context strategies. By comparing the results from these combinations with combination 1, it can be observed that these strategies result in a higher decision consistency. When looking at the individual combinations, combination 3 gave the highest results out of all intra-session strategies. This combination was also the combination with the lowest generation- time and cost.

These findings suggest that selecting an effective intra-session strategy requires high specificity of the saved context. Providing excessive information, as seen in combination 5, did not yield better results, rather it led to a decline in results across most metrics, indicating a point of diminishing returns.

It should, however, be noted that the intra session strategies got worse scores in requirement compliance in comparison to the combinations that use other strategies.

4. **What post-session context management strategies are suitable for summarization, memory extraction, and persistence?**

   Due to the focus shift, the only tested post-session strategy was strategy 3: Session summary. By comparing combination 4 with the baseline found in combination 1, the impact of post-session strategies can be analyzed. The results show a negligible increase in requirement compliance, but with a higher degree of decision compliance. However, this strategy also resulted in a higher generation- time and cost. It can also be observed that combination 4 generated a larger amount of generation errors.

5. **What evaluation criteria and metrics are appropriate to assess continuity?**

   This thesis generally only used one way of assessing continuity, by analyzing how well the different strategies adhered to the different mid-session decisions that were introduced. However, as stated in chapter 5.1, the actual decisions themselves might require more refinement. Despite this, the chosen method remained a valuable tool for measuring continuity when analyzed alongside other collected metrics.

5.3 Additional findings

This thesis did not only answer the problem statements, it also got some additional findings. The results show a difference from the baseline used in combination 1 (using no strategies for context management) and combination 2-5 (using some context management). The results clearly show that for decision consistency, the results were the same or even better in nearly all of the tests for combinations 2 - 5 than from combination 1. The agents ability to remember decisions improved drastically when the agent was asked to save context and read the context before doing the task in that session. With this in mind, a conclusion can be drawn that saving context does make a difference in the agents continuity.

Another conclusion that can be drawn from the results is that it is important to save just enough context and to be specific with what context to save.

In combination 5, when the agent was asked to save a lot of context, the requirement compliance drastically declined, generation- time and costs rose, and decision compliance did not improve significantly compared to the other combinations. Combination 5 also clearly shows an increase in generation errors from all the other combinations. More context contributes to so called "context rot" which can be clearly seen in the results.

The results also indicate that combination 3 achieved a higher decision consistency than the other combinations. It can therefore be interpreted - in regards to continuity - that it is more important to document rules and decisions for future tasks, rather than to document what has been done (To do list and Session summary). Or in other words, strategies that preserve architectural decisions prove to increase continuity more than strategies that store already finished tasks.

5.4 Ethical aspects

The use of AI is a highly ethical dilemma in many aspects. In this chapter, some of the ethical aspects of AI use will be discussed.

5.4.1 Environmental impact

The use of AI has been linked to severe environmental impacts, specifically in regard to the electricity consumption that massive data centers use when training AI models, or just during regular use [29], [30]. Therefore, there are a lot of arguments to be made about whether it is ethically right to use AI on such a large scale. By using AI to produce work that had previously been performed by humans, that usage is actively contributing to higher electricity spending and water usage in areas, leading to increased environmental impacts [31].

5.4.2 Job replacement

In the engineering industry, the use of AI is increasing significantly. More and more people are becoming afraid that AI will replace their jobs. The usage of AI in this thesis has shown that AI has the capability of generating code and results that would previously have to be done by a human, meaning that there is less of a need for programmers at this level. However, studies show that many jobs will not be outright replaced by AI, but rather change to adapt with these newfound tools. Sweden has also been shown to be one of the countries in the world that is best equipped for this transition [26].

5.5 Future development

To further develop this thesis subject, there are a lot of things that can be done. In this thesis, three different context strategies were developed and analyzed. Most of the analysis were on the different strategies, and not on how to make a specific strategy as good as possible. To elaborate further on this point, one should choose

a specific strategy to focus on, and test different forms of context storage methods, different prompts, etc for that strategy.

One could also investigate this thesis area further by focusing more on the original focus of this thesis. This could be done by choosing to investigate a specific part of the context management chain (pre-, intra-, or post-session), and analyzing how different storage methods, retrieval methods, etc. can affect the specific parts of the pipeline.

As discussed in chapter 5.1, the specific wording of the prompts use could have a large impact of the generated results. The actual impact that this has is an area that could be further developed.

Different LLM:s and agentic software may behave very differently, and produce differing results. This thesis only used Codex with GPT5-mini for generation, and the results could have gone differently if other agents and LLM:s were used. Therefore, how the results can differ between models and agents is an area that could also be developed further.

Chapter 6
Terminology

This chapter describes abbreviations and thesis-specific terms and concepts that are not described in chapter 2.

6.1 Thesis-specific terms and concepts

- **Combination:** Refers to a specific combination of context management strategies that were used to generate the mock project. For more information on the mock project's strategy combinations, see chapter 3.3.1.
- **Strategy:** Short for context management strategy. Refers to the context management strategies that were used during generation of the mock project to read, write, and handle context. For more information on what strategies were used during generation of the mock project, see chapter 3.3.
- **Continuity:** The agents ability to maintain consistent understanding of previous decisions and/or constraints.
- **Cycle:** Short for development cycle. Refers to a specific development cycle for the mock project. For more information on the mock project's development cycles, see chapter 3.2.3.
- **Decision:** Short for mid-session decision. Refers to decisions that were made in the middle of a session while generating the mock project. For more information on how the mid-session decisions worked during the generation of the mock project, see chapter 3.2.5.
- **Intra-session:** The middle stage of a session where the agent dynamically retrieves and update context while performing tasks.
- **Mock project:** The project that the agent was asked to create in order to test the different strategies in this thesis. For more information about the mock project, see chapter 3.2.
- **Post-session:** The last stage of a session where the agent stores and summarizes information to use for future sessions.
- **Pre-session:** The first stage of a session where the agent is initialized with relevant background information before the session begins.

- **Requirement:** Refers to a requirement for the mock project. For full requirement specifications see appendix A, figure A.1 and figure A.2.

6.2 Abbreviations

- **AI:** Artificial Intelligence
- **CSV:** Comma Separated Values
- **IDE:** Integrated Development Environment
- **IoT:** Internet of Things
- **KPI:** Key Performance Indicators
- **LLM:** Large Language Machine
- **SDLC:** Software Development Life Cycle

References

[1] S. Peltomaa, "Diagram of Software Development Lifecycle with multiple specialized AI Agents," private communication, 2026.

[2] NASA. "What is Artificial Intelligence?" Accessed: Feb. 11, 2026. [Online]. Available: https://www.nasa.gov/what-is-artificial-intelligence/.

[3] D. Bergmann. "What is Machine Learning?" IBM, Accessed: Feb. 11, 2026. [Online]. Available: https://www.ibm.com/think/topics/machine-learning.

[4] C. Stryker and J. Holdsworth. "What is NLP (natural language processing)?" IBM, Accessed: Feb. 11, 2026. [Online]. Available: https://www.ibm.com/think/topics/natural-language-processing.

[5] IBM. "What are large language models (LLMs)?" Accessed: Feb. 10, 2026. [Online]. Available: https://www.ibm.com/think/topics/large-language-models.

[6] L. Boonstra, "Prompt Engingeering," Feb. 2025. Accessed: Feb. 11, 2026. [Online]. Available: https://archive.org/details/22365-3-prompt-engineering-v-7/mode/2up.

[7] Microsoft. "Understanding tokens," Accessed: Feb. 11, 2026. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/ai/conceptual/understanding-tokens.

[8] P. Rajasekaran, E. Dixon, C. Ryan, and J. Hadfield, "Effective context engineering for AI agents," *Engineering at Anthropic*, Sep. 29, 2025. Accessed: Feb. 11, 2026. [Online]. Available: https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents.

[9] IBM. "What are AI hallucinations?" Accessed: Feb. 11, 2026. [Online]. Available: https://www.ibm.com/think/topics/ai-hallucinations.

[10] D. Bergmann. "What is a context window?" IBM, Accessed: Feb. 11, 2026. [Online]. Available: https://www.ibm.com/think/topics/context-window.

[11] Google Cloud. "What is an AI agent?" Accessed: Feb. 9, 2026. [Online]. Available: https://cloud.google.com/discover/what-are-ai-agents.

[12] IBM. "What are AI Agents?" Accessed: Feb. 9, 2026. [Online]. Available: https://www.ibm.com/think/topics/ai-agents.

[13] OpenAI. "Codex," Accessed: Feb. 9, 2026. [Online]. Available: https://developers.openai.com/codex.

[14] OpenAI. "Quickstart," Accessed: Feb. 9, 2026. [Online]. Available: https://developers.openai.com/codex/quickstart.

[15] OpenAI. "Custom instructions with AGENTS.md," Accessed: Feb. 11, 2026. [Online]. Available: https://developers.openai.com/codex/guides/agents-md/.

[16] LF Projects. "AGENTS.md," Accessed: Feb. 11, 2026. [Online]. Available: https://agents.md/.

[17] Google. "Google NotebookLM," Accessed: Mar. 18, 2026. [Online]. Available: https://notebooklm.google/.

[18] Meta Open Source. "React," Meta, Accessed: Mar. 17, 2026. [Online]. Available: https://react.dev/.

[19] Vite. "Vite | Next Generation Frontend Tooling," VoidZero, Accessed: Mar. 17, 2026. [Online]. Available: https://vite.dev/.

[20] Meta Open Source. "Build a React app from Scratch," Meta, Accessed: Mar. 17, 2026. [Online]. Available: https://react.dev/learn/build-a-react-app-from-scratch.

[21] TypeScript. "TypeScript: JavaScript with syntax for types," Microsoft, Accessed: Mar. 17, 2026. [Online]. Available: https://www.typescriptlang.org/.

[22] Tailwind Labs. "Tailwind CSS - Rapidly build modern websites without ever leaving your HTML," Accessed: Mar. 17, 2026. [Online]. Available: https://tailwindcss.com/.

[23] Recharts Group. "Recharts," Accessed: Mar. 17, 2026. [Online]. Available: https://recharts.github.io/en-US/.

[24] Engineering at Anthropic. "Demystifying evals for AI agents," Anthropic, Accessed: Feb. 25, 2026. [Online]. Available: https://www.anthropic.com/engineering/demystifying-evals-for-ai-agents.

[25] Azure OpenAI. "Azure OpenAI pricing," Accessed: Apr. 22, 2026. [Online]. Available: https://azure.microsoft.com/en-us/pricing/details/azure-openai/.

[26] Sveriges Ingenjörer. "Oron ökar för att AI ska ersätta jobb," Accessed: May 9, 2026. [Online]. Available: https://www.sverigesingenjorer.se/opinion-och-press/nyheter/oron-okar-for-att-ai-ska-ersatta-jobb/.

[27] R. T. McCoy, S. Yao, D. Friedman, M. D. Hardy, and T. L. Griffiths, "Embers of autoregression show how large language models are shaped by the problem they are trained to solve," *Proceedings of the National Academy of Sciences*, vol. 121, no. 41, e2322420121, 2024. DOI: 10.1073/pnas.2322420121. eprint: https://www.pnas.org/doi/pdf/10.1073/pnas.2322420121. [Online]. Available: https://www.pnas.org/doi/abs/10.1073/pnas.2322420121.

[28] A. M. Sharifloo, M. Heydari, P. Kazerooni, D. Maninger, and M. Mezini, "Where do llms still struggle? an in-depth analysis of code generation benchmarks," in *2025 2nd IEEE/ACM International Conference on AI-powered Software (AIware)*, 2025, pp. 249–253. DOI: 10.1109/AIware69974.2025.00035.

[29] O. Håkans, *The environmental impact of using artificial intelligence : Exploring the environmental impact of chatgpt: A literature study and user perception analysis*, 2025. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-361831.

[30] A. S. George, A. S. H. George, and A. S. G. Martin, *The environmental impact of ai: A case study of water consumption by chat gpt*, Apr. 2023. DOI: 10.5281/zenodo.7855594. [Online]. Available: https://www.puiij.com/index.php/research/article/view/39.

[31] R. Turconi, A. Boldrin, and T. Astrup, "Life cycle assessment (lca) of electricity generation technologies: Overview, comparability and limitations," *Renewable and Sustainable Energy Reviews*, vol. 28, pp. 555–565, 2013, ISSN: 1364-0321. DOI: https://doi.org/10.1016/j.rser.2013.08.013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1364032113005534.

Appendix A

Mock project requirement specification documents

This document contains the full requirement specification documents that were used during the generation of the mock project. The project used two requirement documents, one for cycle 1 and an enhanced version for cycle 2. The specifications for cycle 1 can be seen in figure A.1, and for cycle 2 in figure A.2.

```
# Requirements Specification

## 1. Purpose

The system shall provide a local, frontend-only dashboard for
↪  exploring LLM usage via **static mock JSON data stored in
↪  the project repository**, enabling **KPIs**, **timelines**,
↪  **breakdowns**, and **filtering**.

## 2. Scope

### In scope

* React + TypeScript web application built with Vite
* Tailwind CSS styling
* Static mock dataset JSON in the repo
* Data model/types defined in code
* KPI computation and basic charts
* Filtering by user/model/time range and token-type
↪  (input/output/total)
* Events table (inspect raw events)

### Out of scope

* Authentication
* Backend, database, APIs
* Any persistence (no localStorage/IndexedDB)
```

```
* Export functionality

## 3. Target Environment

* Local development only
* Use node.js v18.19.1
* Run command: `npm run dev`

---

# 4. Data Model Requirements

### R-DM-01 - Canonical data model

The system shall define a canonical TypeScript model
↪  `UsageEvent`.

**Required fields:**

* `id: string`
* `timestamp: string` (ISO 8601)
* `user_id: string`
* `model: string`
* `input_tokens: number`
* `output_tokens: number`
* `total_tokens: number`
* `cost_usd: number`
* `status: "success" | "error"`

**Optional fields (recommended):**

* `project: string`
* `environment: "dev" | "prod"`
* `latency_ms: number`

### R-DM-02 - Type safety

All views and computations shall use typed data structures
↪  derived from `UsageEvent`.

### R-DM-03 - Startup validation

The system shall validate the dataset at startup and present a
↪  clear error UI state if invalid.

---
```

```
# 5. Mock Data Requirements

### R-MD-01 - Dataset stored in repo

The system shall include at least one mock dataset JSON file in
↪  the repository (e.g., `src/data/usage-events.json`).

### R-MD-02 - Dataset generator script

The project shall include a script to generate the dataset JSON
↪  file.

**Acceptance criteria:**

* Configurable number of events (e.g., 1k-10k)
* Multiple users and models
* Meaningful time spread (e.g., last 30-90 days)
* Realistic distributions (token variation, some errors)
* Output conforms to `UsageEvent`

---

# 6. Functional Requirements

## 6.1 App shell

### R-UI-01 - Navigation

The system shall provide navigation between:

* Dashboard
* Events

## 6.2 Dashboard

### R-UI-02 - KPIs

The Dashboard shall display for the current filter selection:

* total cost (USD)
* total tokens
* request count
* average cost/request
* average tokens/request

### R-UI-03 - Timeline chart
```

```
The Dashboard shall display a time-series chart that can show:

* cost over time
* tokens over time

### R-UI-04 - Breakdown chart

The Dashboard shall show at least one breakdown visualization:

* cost by model *(preferred)* or tokens by model

## 6.3 Events

### R-EV-01 - Events table

The Events view shall display events in a table with:

* timestamp, user_id, model, input_tokens, output_tokens,
↪  total_tokens, cost_usd, status

### R-EV-02 - Sorting

The table shall support sorting by at least:

* timestamp
* cost_usd
* total_tokens

### R-EV-03 - Event details

Selecting an event shall show full event details (drawer/modal
↪  is fine).

---

# 7. Filtering Requirements

### R-FLT-01 - Shared filters

The system shall provide filters that apply consistently to
↪  both Dashboard and Events.

### R-FLT-02 - Required filters

Filters shall include:
```

```
* time range (preset + custom range OR preset-only if you want
↪  ultra-minimal)
* user_id (single/multi-select)
* model (single/multi-select)
* token metric selector: input/output/total *(affects dashboard
↪  chart and KPI "token" value display)*

### R-FLT-03 - Filter consistency

Filters shall affect:

* KPIs
* timeline chart
* breakdown chart
* events table

---

# 8. Non-Functional Requirements

### R-NFR-01 - No external services

The system shall have no runtime dependency on external
↪  services or APIs.

### R-NFR-02 - Usable with moderate data size

The system shall remain usable with at least ~5,000 events.

### R-NFR-03 - Maintainable structure

Code shall be organized with clear separation (types, data,
↪  aggregation/filter logic, UI).

### R-NFR-04 - Build & run

The project shall run with:

* `npm install`
* `npm run dev`

### R-NFR-05 - Libraries

The project shall use no component libraries

---
```

```
# 9. Deliverables

* Source code repository meeting all requirements above
* Mock dataset JSON committed
* Generator script committed
* Minimal README: run instructions + how to regenerate mock
↪  data + schema reference
```

Figure A.1: Mock project requirements specification for cycle 1

```
# Requirements Specification

## 1. Purpose

The system shall provide a local, frontend-only dashboard for
↪  exploring LLM usage via **static mock JSON data stored in
↪  the project repository**, enabling **KPIs**, **timelines**,
↪  **breakdowns**, and **filtering**.

## 2. Scope

### In scope

* React + TypeScript web application built with Vite
* Tailwind CSS styling
* Static mock dataset JSON in the repo
* Data model/types defined in code
* KPI computation and basic charts
* Filtering by user/model/time range and token-type
↪  (input/output/total)
* Events table (inspect raw events)
* Exporting of Events table based on filters
* Time period comparison

### Out of scope

* Authentication
* Backend, database, APIs
* Any persistence (no localStorage/IndexedDB)

## 3. Target Environment

* Local development only
* Use node.js v18.19.1
* Run command: `npm run dev`
```

```
---

# 4. Data Model Requirements

### R-DM-01 - Canonical data model

The system shall define a canonical TypeScript model
↪  `UsageEvent`.

**Required fields:**

* `id: string`
* `timestamp: string` (ISO 8601)
* `user_id: string`
* `model: string`
* `input_tokens: number`
* `output_tokens: number`
* `total_tokens: number`
* `cost_usd: number`
* `status: "success" | "error"`

**Optional fields (recommended):**

* `project: string`
* `environment: "dev" | "prod"`
* `latency_ms: number`

### R-DM-02 - Type safety

All views and computations shall use typed data structures
↪  derived from `UsageEvent`.

### R-DM-03 - Startup validation

The system shall validate the dataset at startup and present a
↪  clear error UI state if invalid.

---

# 5. Mock Data Requirements

### R-MD-01 - Dataset stored in repo

The system shall include at least one mock dataset JSON file in
↪  the repository (e.g., `src/data/usage-events.json`).
```

```
### R-MD-02 - Dataset generator script

The project shall include a script to generate the dataset JSON
↪  file.

**Acceptance criteria:**

* Configurable number of events (e.g., 1k-10k)
* Multiple users and models
* Meaningful time spread (e.g., last 30-90 days)
* Realistic distributions (token variation, some errors)
* Output conforms to `UsageEvent`

---

# 6. Functional Requirements

## 6.1 App shell

### R-UI-01 - Navigation

The system shall provide navigation between:

* Dashboard
* Events
* Comparison

## 6.2 Dashboard

### R-UI-02 - KPIs

The Dashboard shall display for the current filter selection:

* total cost (USD)
* total tokens
* request count
* average cost/request
* average tokens/request

### R-UI-03 - Timeline chart

The Dashboard shall display a time-series chart that can show:

* cost over time
* tokens over time

### R-UI-04 - Breakdown chart
```

```
The Dashboard shall show at least one breakdown visualization:

* cost by model *(preferred)* or tokens by model

### R-UI-05 - Heatmap chart

The dashboard shall show a calendar heatmap chart in accordance
↪  to the requirements in 6.3

## 6.3 Heatmap

### R-HMP-01 - Metrics

The heatmap chart shall showcase the amount of events per day
↪  over the last year

### R-HMP-02 - Year selection

The heatmap chart shall showcase events from a selectable year

### R-HMP-03 - Filters

The heatmap chart shall ignore the time-range filter, but shall
↪  apply the other currently selected filters

### R-HMP-04 - Legend

The heatmap chart shall display a legend that shows the current
↪  color thresholds

### R-HMP-05 - Thresholds

The heatmap charts color thresholds shall adapt to the current
↪  values to ensure color clarity

### R-HMP-06 - Labels

The heatmap chart shall show labels for weekdays and months

## 6.4 Events

### R-EV-01 - Events table

The Events view shall display events in a table with:
```

```
* timestamp, user_id, model, input_tokens, output_tokens,
↪  total_tokens, cost_usd, status

### R-EV-02 - Sorting

The table shall support sorting by at least:

* timestamp
* cost_usd
* total_tokens

### R-EV-03 - Event details

Selecting an event shall show full event details (drawer/modal
↪  is fine).

### R-EV-04 - Event table pagination

The Events table shall use pagination with a page size of 50
↪  events

## 6.5 Comparison

### R-CO-01 - Time Periods

The Comparison page shall support selection of two different
↪  time periods

### R-CO-02 - Time period selection

Time period selection for comparison shall be done by selecting
↪  start- and end dates

### R-CO-03 - Compare

The Comparison page shall display for the current filters a
↪  comparison in KPIs between the two selected time periods

### R-CO-04 - KPIs

The Comparison page shall use the same KPIs as requirement
↪  R-UI-02

### R-CO-05 - Highlights

The Comparison page shall highlight the difference in KPIs for
↪  the selected time periods
```

```
---

# 7. Exporting Requirements

### R-EX-01 - Export event table

The Events table shall be able to be exported

### R-EX-02 - Export file Format

The exported file shall be in the `.csv` format

### R-EX-03 - Export filters

The exported events shall respect the currently selected event
↪  filters

### R-EX-04 - Export sorting

The exported events shall respect the currently selected table
↪  sorting

---

# 8. Filtering Requirements

### R-FLT-01 - Shared filters

The system shall provide filters that apply consistently to
↪  both Dashboard, Events and Comparison.

### R-FLT-02 - Required filters

Filters shall include:

* time range (preset + custom range OR preset-only if you want
↪  ultra-minimal)
* user_id (single/multi-select)
* model (single/multi-select)
* token metric selector: input/output/total *(affects dashboard
↪  chart and KPI ‘‘token’’ value display)

### R-FLT-03 - Filter consistency

Filters shall affect:
```

```
* KPIs
* timeline chart
* breakdown chart
* heatmap chart
* events table

---

# 9. Non-Functional Requirements

### R-NFR-01 - No external services

The system shall have no runtime dependency on external
↪  services or APIs.

### R-NFR-02 - Usable with moderate data size

The system shall remain usable with at least ~5,000 events.

### R-NFR-03 - Maintainable structure

Code shall be organized with clear separation (types, data,
↪  aggregation/filter logic, UI).

### R-NFR-04 - Build & run

The project shall run with:

* `npm install`
* `npm run dev`

### R-NFR-05 - Libraries

The project shall use the following Node.js libraries:

* Recharts for graphs

---

# 10. Deliverables

* Source code repository meeting all requirements above
* Mock dataset JSON committed
* Generator script committed
* Minimal README: run instructions + how to regenerate mock
↪  data + schema reference
```

Figure A.2: Mock project requirements specification for cycle 2

Appendix B

Mock project session breakdown

This document contains breakdowns of the generation of the mock project: what should be generated and during which session and cycle. The session breakdown for cycle 1 can be seen in figure B.1, and for cycle 2 in figure B.2.

```
# Agent CLI Session Plan Cycle 1

## Session 1 - Scaffold + app shell

* Create Vite React TS project
* Add Tailwind
* Add routing and pages: Dashboard, Events
* Basic layout (nav/header)
  **Covers:** R-UI-01, R-NFR-04, R-NFR-03

## Session 2 - UsageEvent data model + loader + validation

* Implement `UsageEvent` type
* Load JSON from repo
* Validate on startup + error state UI
  **Covers:** R-DM-01..03, R-MD-01

## Session 3 - Mock data generator

* Implement generator script (configurable size/time
↪  range/users/models)
* Generate `usage-events.json` and commit
* Add README section for regeneration
  **Covers:** R-MD-02, Deliverables

## Session 4 - Events table + details

* Events table with sorting
* Row click → details drawer/modal
  **Covers:** R-EV-01..03
```

```
## Session 5 - Filtering + aggregations + KPIs

* Implement shared filter state (time/user/model/token-type)
* Apply filters to dataset
* Compute KPIs from filtered set
* Render KPI cards
  **Covers:** R-FLT-01..03, R-UI-02

## Session 6 - Charts (timeline + breakdown)

* Time bucketing and timeline chart for cost/tokens
* Breakdown chart (cost by model)
* Token metric selector affects chart display
  **Covers:** R-UI-03, R-UI-04, R-FLT-02
```

Figure B.1: Session breakdown for cycle 1

```
# Agent CLI Session Plan Cycle 2

## Session 1 - Refactor and improve existing features (charts
↪  library + pagination)

* Add Recharts library in the project
* Refactor the charts in the project to use the Recharts
↪  library
* Implement pagination on the events table
  **Covers:** R-EV-04, R-NFR-05

## Session 2 - Export events table

* Implement export functionality of events table
* Exports to csv file
* Export shall use selected filters and sorting
  **Covers:** R-EX-01..04

## Session 3 - Period comparison

* Create new page for comparison view
* Add routing and navigation to comparison view
* Implement comparison page to compare KPIs of two time periods
  **Covers:** R-FLT-01, R-UI-01, R-CO-01..04

## Session 4 - Calendar heatmap
```

```
* Implement a calendar heatmap over the events on Dashboard
* Use selected filters
* Use a suitable library for the heatmap chart
  **Covers:** R-UI-05, R-FLT-03, R-HMP-01..06
```

Figure B.2: Session breakdown for cycle 2

Appendix C

Mock project prompts

This document contains the actual prompts that were inputted to the agent in order to generate the mock project. The prompts used in cycle 1 can be seen in figure C.1, and for cycle 2 in figure C.2.

```
# Session 1

## Prompt 1 (Main content)
Goal:
Create the scaffold + app shell

Context:
Page contents will be added later, implement only scaffolding

Instructions:
1. Create a Vite React Typescript project with Tailwind
2. Implement the following routes and empty pages:
   - Dashboard
   - Events
3. Add a basic layout with a navbar and a header

Do the instructions step by step

## Prompt 2 (Mid-session decision)
I want all headings in the app to have the text "(Heading)"
↪  appended at the end of the content of each heading tag.
↪  Example: <h1>Welcome (Heading)</h1>. This must be done
↪  manually for every heading, no automation.

---

# Session 2

## Prompt 1 (Mid-session decision)
```

```
I want all `.tsx` and `.ts` files to have a comment at the top
↪  of the files containing the date and time the file was
↪  created. For already existing files, use the current date
↪  and time for created. This must be done manually, no
↪  automation.

## Prompt 2 (Main content)
Goal:
Create a data model + loader + validation

Context:
Mock data will be generated later, only implement the currently
↪  specified features

Instructions:
1. Implement the UsageEvent type in accordance to R-DM-01
2. Add the ability to load the mock json dataset from the repo
3. Add startup validation and error state UI

Do the instructions step by step.

---

# Session 3

## Prompt 1 (Main content)
Goal:
Create the mock data generator

Instructions:
1. Implement the generator script with configurable
↪  size/time/range/users/models
2. Generate the mock JSON data usage-events.json
3. Add a README section for regeneration


Do the instructions step by step.

## Prompt 2 (Mid-session decision)
I want every model used in the mock data to be displayed in the
↪  app with its own specified color. When the name of a model
↪  is displayed, it should be colored with its own color.

Pick a color palette where every model is represented by a
↪  unique color, and save it in a configuration file.

---
```

```
# Session 4

## Prompt 1 (Main content)
Goal:
Create events table + details

Instructions:
1. Implement events table on the Events page with
↪  column-sorting functionality
2. When clicking on a row, open a modal or drawer showcasing
↪  full event details about that event

Do the instructions step by step.

## Prompt 2 (Mid-session decision)
I want all new pages pages to be placed inside a
↪  `pages/`-folder for better structure. Don't move existing
↪  pages.

---

# Session 5
Goal:
Create the filtering + aggregations + KPIs

Instructions:
1. Implement a shared filter state in accordance to the filter
↪  requirements
2. Ensure that the filter selection component is displayed on
↪  all pages where it is applied
3. Add the ability to apply the filters to a dataset and
↪  compute KPIs from the filtered set
4. Render the KPI cards

Do the instructions step by step.

---

# Session 6
Goal:
Create the charts (timeline + breakdown)

Instructions:
1. Implement time bucketing and timeline charts for either cost
↪  and tokens, selectable by the user
2. Add a breakdown chart showcasing cost by model
```

```
3. Add the ability for the token metric selector to affect the
↪  charts display

Do the instructions step by step.
```

Figure C.1: Generation prompts for cycle 1

```
# Session 1
Goal:
Improve already existing features

Instructions:
1. Add the Recharts library to the project
2. Refactor the charts in the Dashboard to use the Recharts
↪  library
3. Implement pagination on the events table in accordance to
↪  R-EV-04

Do the instructions step by step

---

# Session 2
Goal:
Create functionality to export events table

Context:
The user needs to be able to take their filtered and sorted
↪  data out of the app for external analysis

Instructions:
1. Add an Export to CSV button on the Events page
2. Implement logic to generate a `.csv` file from the current
↪  table state in accordance to the export requirements

Do the instructions step by step.

---

# Session 3
Goal:
Build a comparison page to analyze usage trends across two time
↪  periods

Instructions:
```

```
1. Create a new page for comparison
2. Add routing and navigation to comparison
3. Implement the two separate time period selectors
4. Display a comparison view for the KPIs of the selected time
↪  periods in accordance to R-CO-01..05

Do the instructions step by step.

# Session 4
Goal:
Implement a calendar heatmap chart over the events on Dashboard
↪  to visualize usage frequency

Instructions:
1. Implement a calendar heatmap component in accordance to
↪  requirements R-HMP-01..06
2. Render the heatmap to the Dashboard to show events per day
↪  over the last year
3. Add a year selector
4. Ensure that the requirements R-UI-05 and R-HMP-01..06 is
↪  followed

Do the instructions step by step.
```

Figure C.2: Generation prompts for cycle 2

Appendix D

AGENTS.md files

This document contains all `AGENTS.md` files that were used in order to inject instructions and context management strategies to the agent during generation of the mock project. All combinations of context management strategies can be found in the following figures:

1. **No strategies:** Figure D.1.
2. **Only (1) to-do list:** Figure D.2
3. **Only (2) decision log:** Figure D.3
4. **Only (3) session summary:** Figure D.4
5. **All three strategies (1, 2, 3):** Figure D.5

```
# LLM Dashboard
You are creating a web app containing a dashboard for exploring
↪  LLM usage

## Compatibility
- Use node.js v18.19.1
- Use Vite v5.4.11
- use Tailwind v3.4.13

## Structure
- requirements.md: Markdown file that contains the requirements
↪  and scope for the project
- agent-context/: Folder where all context files shall be
↪  stored

## Rules
- For every feature and implementation, check requirements.md
↪  to view requirements for that feature and its environment
- When installing any node modules, always make sure that the
↪  version is compatible with your environment and versions
- Never commit anything to git
```

```
- Never start the dev server, but always verify that the build
↪  works, especially when adding new libraries

## Context management rules
Don't save any additional context.
```

Figure D.1: Full AGENTS.md for combination 1

```
# LLM Dashboard
You are creating a web app containing a dashboard for exploring
↪  LLM usage

## Compatibility
- Use node.js v18.19.1
- Use Vite v5.4.11
- use Tailwind v3.4.13

## Structure
- requirements.md: Markdown file that contains the requirements
↪  and scope for the project
- agent-context/: Folder where all context files shall be
↪  stored

## Rules
- For every feature and implementation, check requirements.md
↪  to view requirements for that feature and its environment
- When installing any node modules, always make sure that the
↪  version is compatible with your environment and versions
- Never commit anything to git
- Never start the dev server, but always verify that the build
↪  works, especially when adding new libraries

## Context management rules
### Todo list
Before executing any code or modifications, you must initialize
↪  the task context to ensure decisions are documented and
↪  previous logic is reused to maintain continuity.

#### Phase 1: Context Retrieval
1. Search the `agent-context/todo/` directory for existing JSON
↪  context files.
2. Identify patterns, previous architectural decisions, or
↪  completed steps that impact the current task.
3. Explicitly acknowledge any relevant previous context before
↪  proceeding.
```

```
#### Phase 2: Planning & Documentation
When a new task is assigned, you shall create a structured
↪  breakdown before writing any functional code:
1. Generate a new JSON file at
↪  `agent-context/todo/{task_name}.json`.
2. Use the following structure for the JSON:
   - `task`: A short text describing the task and its purpose
   - `steps`: A list of objects containing `step`, `status`
   ↪  (pending/done) and `notes` (short comment on what was
   ↪  done and if applicable, how it will impact future tasks)

#### Phase 3: Execution & Sync
1. Update the `status` and `notes` within the JSON file
↪  immediately as you progress through or complete a step.
2. Ensure the JSON file reflects the "ground truth" of the
↪  task's current state at all times.
```

Figure D.2: Full AGENTS.md for combination 2

```
# LLM Dashboard
You are creating a web app containing a dashboard for exploring
↪  LLM usage

## Compatibility
- Use node.js v18.19.1
- Use Vite v5.4.11
- use Tailwind v3.4.13

## Structure
- requirements.md: Markdown file that contains the requirements
↪  and scope for the project
- agent-context/: Folder where all context files shall be
↪  stored

## Rules
- For every feature and implementation, check requirements.md
↪  to view requirements for that feature and its environment
- When installing any node modules, always make sure that the
↪  version is compatible with your environment and versions
- Never commit anything to git
- Never start the dev server, but always verify that the build
↪  works, especially when adding new libraries

## Context management rules
```

```
### Decision logging
Before executing any code or modifications, you must consult
↪  the persistent decision log to ensure alignment with
↪  established patterns. Every significant technical choice
↪  made during this session must be recorded for future
↪  context to maintain continuity.

#### Phase 1: Context retrieval
1. Scan `agent-context/decisions/` for all existing JSON
↪  context files
2. Identify any specific architectural constraints, patterns,
↪  or logical decisions that apply to the current request

#### Phase 2: Decision capture
Whenever a decision, rule or design choice is made by the user,
↪  document it immediately
1. Create or update a JSON file at
↪  `agent-context/decisions/{feature_area}.json`
2. Use the following structure for the JSON:
   - `decision`: Short name for the decision
   - `description`: Describe the decision in required level of
   ↪  detail

#### Phase 3: Session summary
At the end of a task, verify that the
↪  `agent-context/decisions/` folder accurately reflects the
↪  current state of the application's architecture
```

Figure D.3: Full AGENTS.md for combination 3

```
# LLM Dashboard
You are creating a web app containing a dashboard for exploring
↪  LLM usage

## Compatibility
- Use node.js v18.19.1
- Use Vite v5.4.11
- use Tailwind v3.4.13

## Structure
- requirements.md: Markdown file that contains the requirements
↪  and scope for the project
- agent-context/: Folder where all context files shall be
↪  stored
```

```
## Rules
- For every feature and implementation, check requirements.md
↪  to view requirements for that feature and its environment
- When installing any node modules, always make sure that the
↪  version is compatible with your environment and versions
- Never commit anything to git
- Never start the dev server, but always verify that the build
↪  works, especially when adding new libraries

## Context management rules
### Session summarization
Before beginning any work you must review the history of
↪  previous sessions to maintain continuity. At the conclusion
↪  of the current session (when the user writes `EXIT
↪  SESSION`), you must write down all actions into a concise
↪  summary for future use.

#### Phase 1: Context reconstruction
1. Search the `agent-context/sessions/` directory for existing
↪  JSON summary files.
2. Review the `summary` and `pending_threads` from the previous
↪  session to understand the immediate starting point.

#### Phase 2: Activity tracking
Throughout the session, maintain an internal log of:
1. Significant file changes or new components created.
2. Problems encountered and how they were addressed.
3. Specific user preferences or instructions shared during the
↪  dialogue.

#### Phase 3: Handoff generation
When the user types `EXIT SESSION`, you must create a JSON file
↪  at `agent-context/sessions/session_{timestamp}.json` using
↪  the following structure:
- `summary`: A high-level overview of what was achieved during
↪  this session
- `changes`: A list of files modified
- `pending_threads`: Specific tasks or unresolved logic for
↪  future sessions to address.
```

Figure D.4: Full AGENTS.md for combination 4

```
# LLM Dashboard
You are creating a web app containing a dashboard for exploring
↪  LLM usage
```

```
## Compatibility
- Use node.js v18.19.1
- Use Vite v5.4.11
- use Tailwind v3.4.13

## Structure
- requirements.md: Markdown file that contains the requirements
↪  and scope for the project
- agent-context/: Folder where all context files shall be
↪  stored

## Rules
- For every feature and implementation, check requirements.md
↪  to view requirements for that feature and its environment
- When installing any node modules, always make sure that the
↪  version is compatible with your environment and versions
- Never commit anything to git
- Never start the dev server, but always verify that the build
↪  works, especially when adding new libraries

## Context management rules
### Todo list
Before executing any code or modifications, you must initialize
↪  the task context to ensure decisions are documented and
↪  previous logic is reused to maintain continuity.

#### Phase 1: Context Retrieval
1. Search the `agent-context/todo/` directory for existing JSON
↪  context files.
2. Identify patterns, previous architectural decisions, or
↪  completed steps that impact the current task.
3. Explicitly acknowledge any relevant previous context before
↪  proceeding.

#### Phase 2: Planning & Documentation
When a new task is assigned, you shall create a structured
↪  breakdown before writing any functional code:
1. Generate a new JSON file at
↪  `agent-context/todo/{task_name}.json`.
2. Use the following structure for the JSON:
   - `task`: A short text describing the task and its purpose
   - `steps`: A list of objects containing `step`, `status`
   ↪  (pending/done) and `notes` (short comment on what was
   ↪  done and if applicable, how it will impact future tasks)

#### Phase 3: Execution & Sync
```

```
1. Update the `status` and `notes` within the JSON file
↪  immediately as you progress through or complete a step.
2. Ensure the JSON file reflects the "ground truth" of the
↪  task's current state at all times.

### Decision logging
Before executing any code or modifications, you must consult
↪  the persistent decision log to ensure alignment with
↪  established patterns. Every significant technical choice
↪  made during this session must be recorded for future
↪  context to maintain continuity.

#### Phase 1: Context retrieval
1. Scan `agent-context/decisions/` for all existing JSON
↪  context files
2. Identify any specific architectural constraints, patterns,
↪  or logical decisions that apply to the current request

#### Phase 2: Decision capture
Whenever a decision, rule or design choice is made by the user,
↪  document it immediately
1. Create or update a JSON file at
↪  `agent-context/decisions/{feature_area}.json`
2. Use the following structure for the JSON:
   - `decision`: Short name for the decision
   - `description`: Describe the decision in required level of
   ↪  detail

#### Phase 3: Session summary
At the end of a task, verify that the
↪  `agent-context/decisions/` folder accurately reflects the
↪  current state of the application's architecture

### Session summarization
Before beginning any work you must review the history of
↪  previous sessions to maintain continuity. At the conclusion
↪  of the current session (when the user writes `EXIT
↪  SESSION`), you must write down all actions into a concise
↪  summary for future use.

#### Phase 1: Context reconstruction
1. Search the `agent-context/sessions/` directory for existing
↪  JSON summary files.
2. Review the `summary` and `pending_threads` from the previous
↪  session to understand the immediate starting point.

#### Phase 2: Activity tracking
```

```
Throughout the session, maintain an internal log of:
1. Significant file changes or new components created.
2. Problems encountered and how they were addressed.
3. Specific user preferences or instructions shared during the
↪  dialogue.

#### Phase 3: Handoff generation
When the user types `EXIT SESSION`, you must create a JSON file
↪  at `agent-context/sessions/session_{timestamp}.json` using
↪  the following structure:
- `summary`: A high-level overview of what was achieved during
↪  this session
- `changes`: A list of files modified
- `pending_threads`: Specific tasks or unresolved logic for
↪  future sessions to address.
```

Figure D.5: Full AGENTS.md for combination 5