

ScheduleMe: Multi-Agent Calendar Assistant

Oshadha Wijerathne¹, Amandi Nimasha¹, Dushan Fernando¹,
Nisansa de Silva¹, and Srinath Perera²

¹Department of Computer Science & Engineering, University of Moratuwa, Sri Lanka
{oshadha.20, amandi.20, dushan.20, NisansaDdS}@cse.mrt.ac.lk

²WSO2 LLC
srinath@wso2.com

Abstract

Recent advancements in LLMs have contributed to the rise of advanced conversational assistants that can assist with user needs through natural language conversation. This paper presents a *ScheduleMe*, a multi-agent calendar assistant for users to manage google calendar events in natural language. The system uses a graph-structured coordination mechanism where a central supervisory agent supervises specialized task agents, allowing modularity, conflicts resolution, and context-aware interactions to resolve ambiguities and evaluate user commands. This approach sets an example of how structured reasoning and agent cooperation might convince operators to increase the usability and flexibility of personal calendar assistant tools.

1 Introduction

The rapid advancements in natural language processing (NLP) and large language models (LLMs) have opened new opportunities for developing intelligent, user-friendly applications. Among these, conversational agents capable of understanding and acting upon human language inputs are becoming increasingly important for daily task management. Traditional scheduling systems often require rigid, form-based inputs and manual navigation, limiting user experience and efficiency. In contrast, leveraging LLMs enables the creation of systems that can interpret flexible, natural language instructions, offering a more intuitive and seamless interaction.

This research introduces a calendar management assistant built using the LangChain framework (Følstad and Skjuve, 2019) and OpenAI’s GPT-4o mini model (OpenAI et al., 2024; Wang, 2025). The assistant enables users to manage their calendars through natural, conversational interactions.

The system adopts a modular, multi-agent architecture. Specialized agents handle specific operations such as scheduling, fetching, editing, and

deleting events. A centralized supervisory chatbot coordinates these agents and manages the dialogue with the user. This separation improves modularity, reliability, and context-aware task execution.

The goal is to create an assistant that not only executes tasks accurately based on natural user requests, but also enhances transparency, reliability, and user satisfaction. Users will be able to manage their calendars simply by conversing with the chatbot, eliminating the need for complex interfaces or strict command formats. Through this research, our objective is to demonstrate how LLMs, when properly structured within a robust framework such as LangChain, can serve as powerful tools for building practical, real-world intelligent applications that align with human communication patterns.

2 Related Work

2.1 Limitations of Traditional Scheduling Dialogue Systems

Traditional task-oriented dialogue systems typically rely on intent classification and slot-filling methods (Surdeanu et al., 2011). These systems map user inputs to predefined actions and extract key details such as date, time, or participants.

Dialogue flows are often rigid and frame-based (Braggaar et al., 2024), collecting user input step-by-step. While effective in simple cases, this structure struggles when users provide information out of order, revise commands, or use unexpected phrasing. Systems such as *Calendar.help* (Cranshaw et al., 2017) have shown how these methods can be used in real-world scheduling applications, combining natural language understanding with backend tools such as webhook integrations. However, these systems often fail to handle ambiguous or incomplete inputs gracefully and are difficult to adapt to new use cases without retraining on labeled data. As a result, their user interactions can feel rigid and frustrating.

2.2 Advancing to LLM-Based Multi-Agent Systems

Earlier AI agents were designed using symbolic rules or simple learning methods were built to act independently, make decisions, and sometimes communicate with other agents (Farinetti and Canale, 2024; Alonso, 2002; Hazra et al., 2024). Although these early agents were effective in narrow tasks, they required a lot of manual programming and were not very flexible (Wang et al., 2024).

Recent advances in large language models (LLMs), such as GPT-4 (OpenAI et al., 2024; Qin et al., 2023), PaLM (Chowdhery et al., 2023), and LLaMA (Touvron et al., 2023), have introduced new possibilities. These models can handle open-ended language tasks, reason about goals, and even use external tools when guided properly. Methods such as chain-of-thought prompting (Wei et al., 2022), in-context learning, and tool use through APIs have enabled LLM-based agents to solve more complex and varied problems (Warnakulasuriya et al., 2025).

Frameworks such as ReAct (Yao et al., 2023) and AutoGPT (Yang et al., 2023) have demonstrated how LLMs can be used as the 'brains' behind autonomous agents. However, many tasks such as scheduling or workflow management require multiple specialized agents to work together. A multi-agent setup allows for modular design, parallel task execution, and clearer delegation of responsibilities. Without structured coordination, these systems often struggle with effective communication, shared memory, and maintaining context, which limits their ability to handle complex or extended interactions.

2.3 Multi-Agent Approaches to Scheduling Assistants

Some recent systems split scheduling tasks into smaller parts, assigning different agents to handle event creation, editing, retrieval, and similar operations. *SmartCal* (Shen et al., 2024) improves tool use reliability and decision making through a self-aware supervisory framework, which coordinates agent actions, supports error recovery, and maintains context. Without such supervision, assistants often struggle with user corrections and overlapping goals during complex interactions.

Beyond orchestration, procedural and multi-step reasoning is crucial for advanced scheduling. ScriptWorld (Joshi et al., 2023) demonstrates how

agents can learn and execute sequential tasks while maintaining state. Although it operates in simulation, the same principle applies to real-world scheduling, where agents must interpret goals, resolve conflicts, and complete tasks in the correct order.

To overcome the limits of static coordination, frameworks like LangGraph (Duan and Wang, 2024; Wang and Duan, 2024) provide graph-based, state-aware workflows. Nodes represent task-specific agents or tools, and edges define transitions based on system state, enabling conditional branching, iteration, and runtime adaptation. Integrating LangGraph with a central supervisory agent supports dynamic task execution, context maintenance, and natural conversational scheduling.

3 Methodology

3.1 Multi-Agent System Architecture

Having discussed prior work and current limitations in existing systems, we now present the architecture and design principles behind our proposed calendar assistant. Our system integrates large language model (LLM) based reasoning with a graph-driven orchestration framework using LangGraph, enabling dynamic coordination among agents.

The architecture is centered around a supervisory chatbot agent, which serves as the sole interface for user interaction and agent coordination. Upon receiving a user query, the supervisor agent interprets user intent and delegates tasks accordingly to one of the specialized functional agents. These include the scheduling agent, availability checking agent, event editing agent, and event deletion agent. All inter-agent communication is mediated through the supervisor agent. When a functional agent requires additional information to complete a task, it requests the chatbot to re-engage with the user to obtain the missing input.

Each agent follows the ReAct (Reasoning and Acting) paradigm, combining decision-making with the ability to invoke predefined tools. These tools are implemented as custom functions that interface with the Google Calendar API to perform specific actions. For instance, the availability checking tool queries calendar data for events within a given time range, while the scheduling tool creates new calendar events based on parameters such as title and time. Similarly, the editing and deletion tools update or remove events based on event ids and user-specified criteria. These tools

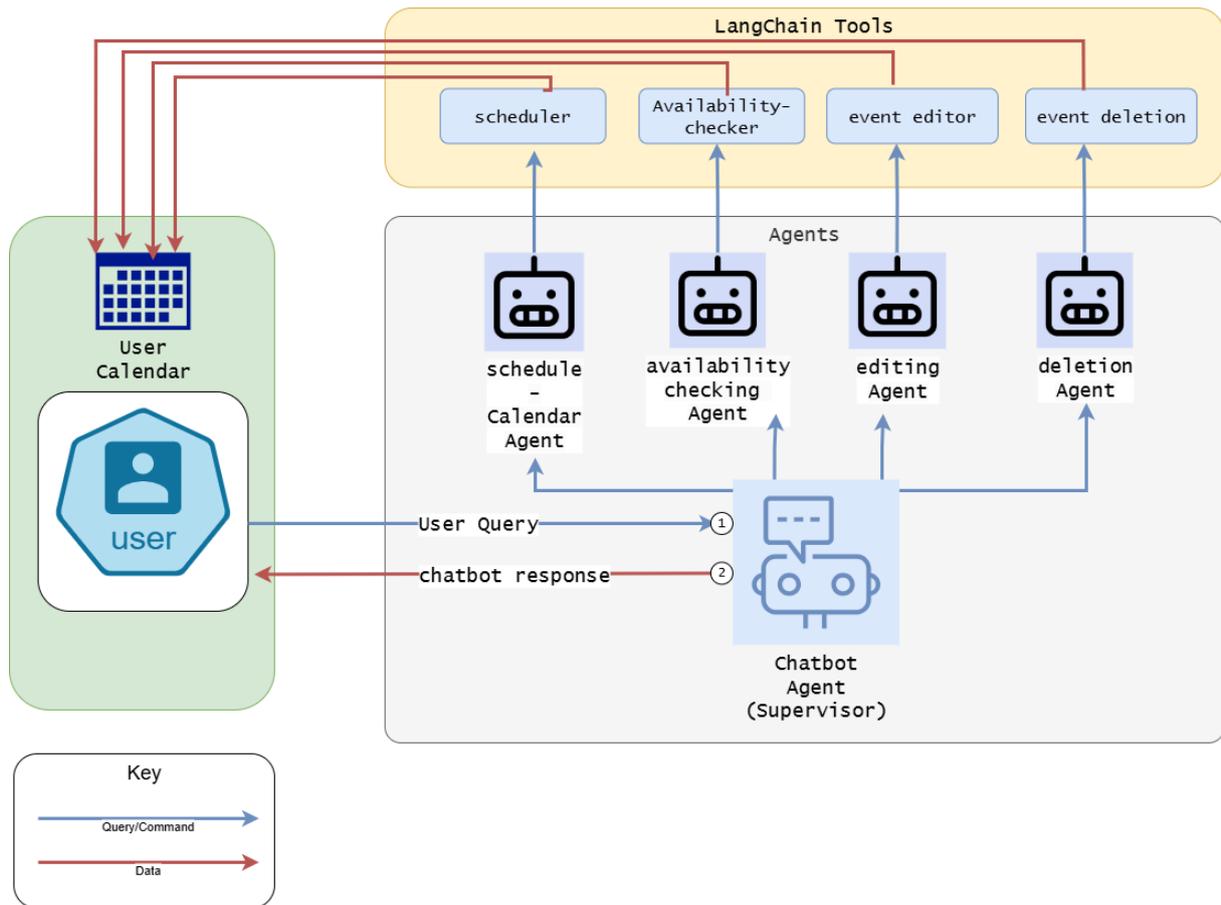


Figure 1: Multi-Agent System Architecture: All other agents are controlled by the *supervisor agent*, but we have opted not to draw the control and communication lines between the agents to reduce unnecessary clutter. When a command or a data item is relevant to all the entities in a parent entity, the relevant arrow terminates on the parent entity. Otherwise, it terminates on the specific relevant child entity. The numbers on the agents at arrow terminals indicate the order in which each action may happen in a typical execution.

abstract the underlying API calls, allowing agents to focus on high-level decision logic.

LangGraph is employed to structure the agent coordination process as a directed graph. In this configuration, each node represents an agent, and edges define the flow of execution based on the outcome of reasoning steps or user input. The graph structure enforces that communication paths flow through the supervisor agent, ensuring a controlled and interpretable interaction model. This setup allows the system to flexibly handle user queries in a stateful and modular manner. An overview of the system’s architecture is depicted in Fig. 1, highlighting the agents and their interactions within the LangGraph execution framework.

3.2 Implementation Details

With the system architecture defined, we now describe how the AI Calendar Assistant is implemented in practice, detailing the technologies and

components involved. The AI Calendar Assistant is implemented as a graph-based multi-agent system, where each node corresponds to an agent and each edge represents a possible transition in the task flow. The architecture is constructed using LangGraph’s StateGraph module, which supports dynamic, stateful execution paths. The central supervisory chatbot agent initializes each interaction by processing user input and extracting intent, parameters, and task directives. These outputs determine the subsequent traversal of the graph and the activation of the appropriate functional agent.

The system integrates OpenAI’s GPT-4o mini model via LangChain to perform natural language understanding and dialogue management. GPT-4 is known for high accuracy in complex reasoning but is resource-intensive and costly (Gunathilaka and de Silva, 2025; Siddiky et al., 2025). GPT-4o provides comparable performance, with stronger multi-lingual and multimodal capabilities, while offering

reduced latency and computational requirements (Siddiky et al., 2025; Zhang et al., 2024). Prior studies in dialogue system design have also employed GPT-4o mini as a reference model due to its extended (128k-token) context window, function-calling support, and low latency (Robino, 2025). In the medical domain, GPT-4o has demonstrated efficiency gains and near-human conversational response times, confirming its suitability for real-time applications (Zhang et al., 2024). The supervisor agent leverages this model to interpret user queries, ask for clarification when needed, and generate tailored instructions for each functional agent. The initiation of the supervisor is shown in Prompt 1. When activated, functional agents handle tasks such as scheduling, checking availability, editing events, or deleting them. These agents perform stateless operations: they receive a structured input payload, carry out the task, and return the results to the supervisor. The supervisor then communicates the outcome back to the user. We show all the functional agent prompts in Appendix A.

```

"""You are the Supervisor Agent for an AI
Calendar Assistant system.

Current date and time: {current_date_time}.

Your Responsibilities:
- Talk to the user to fully understand their
  request.
- Collect all required information
  before sending a task to any agent.
- Send tasks to the correct agent with
  complete and clear information.
- Collect responses from agents and decide
  the next action.

Agents you can use:
- calendar_checker_agent: To check calendar
  events.
- event_scheduler_agent: To add new events (
  REQUIRES: event title, date, and time).
- event_remover_agent: To delete events.(
  Should Provide the event Id.)
- event_modifier_agent: To modify/edit/
  update events.
- user: If you need more information.

Important Rules:
1. Greet the user and ask what they want to
  do.
2. If user request is unclear or missing
  information, ask follow-up questions (
  one at a time) until you have
  everything needed.
3. Only send a task to another agent once
  you have all required information.
4. Be friendly, clear, and simple. Ask one
  question at a time.
5. Always format your reply in JSON:
  - 'next': agent to call ('
    calendar_checker_agent', '
    event_scheduler_agent', '
    event_editor_agent', 'user', or '
    FINISH')
  - 'messages': Message content (talk to
    the user or explain to the agent
    what task to do).

```

```

**EXTRA REMINDERS:**
- For scheduling an event: you must collect
  event title, date, and time
**.
- For deleting: you must collect event_ID
**.
- For editing : you must collect event
  title and what exactly to edit.
- If something is unclear, always ask the
  user first instead of guessing.

Example JSON message when enough info is
collected:
```json

 "next": "event_scheduler_agent",
 "messages": "Schedule an event titled '
 Team Meeting' on 2025-05-01 at 10:00
 AM."

Prompt 1: Chat-bot Supervisor Prompt

```

The backend infrastructure is developed using FastAPI, which provides RESTful endpoints for user interaction, session handling, and communication with the agent graph. A persistent JSON-based state file maintains context across multi-turn interactions, enabling coherent task handling. Secure integration with Google Calendar is achieved via OAuth 2.0 authentication, allowing the system to create, retrieve, modify, and delete calendar events with user consent.

Temporal data handling is managed using libraries such as pytz for timezone resolution and dateparser for parsing natural language dates and times. Each task agent is equipped with a custom tool that wraps the corresponding Google Calendar API call, encapsulating the logic required for scheduling, availability queries, editing, or deletion. These tools enforce structured data handling and reduce coupling between agents and external APIs.

A frontend interface for the assistant is developed using Streamlit, allowing users to interact with the calendar assistant through a simple web-based UI. The interface displays an updated calendar view on the right side of the page, providing users with a visual confirmation of their calendar events and any modifications made by the assistant.

Figure 2 shows a representative interaction example with the AI Calendar Assistant. The chosen stack LangGraph, LangChain, GPT-4o mini, FastAPI, Google Calendar API, and Streamlit provides a scalable, modular, and interpretable framework for enabling intelligent calendar management through natural language conversations.

## AI Calendar Assistant

 Hi there! I'm your AI Calendar Assistant. I can help you check, schedule, edit, or delete events from your calendar. How can I assist you today?

 Can you check my events for tomorrow?

 chatbot\_supervisor : Check for any events scheduled for May 4, 2025.

 calendar\_checker\_agent : On May 4, 2025, the user has the following scheduled events:

1. **Event:** Lunch with Alex  
**Time:** 12:30 PM - 1:30 PM (Asia/Colombo)  
**Event ID:** i9ucsc15hmumlfjr2q45Sanjk4  
[View Event](#)  
**Description:** Catching up over lunch at Green Leaf Café.
2. **Event:** Doctor's Appointment  
**Time:** 3:00 PM - 4:00 PM (Asia/Colombo)  
**Event ID:** m1jji6h33hsu3349ike1equkrs  
[View Event](#)

The user has prior commitments during this day, and therefore may not be available. Would you like me to check for another date?

 chatbot\_supervisor : You have two events scheduled for tomorrow, May 4, 2025:

1. **Lunch with Alex** from 12:30 PM to 1:30 PM.
2. **Doctor's Appointment** from 3:00 PM to 4:00 PM.

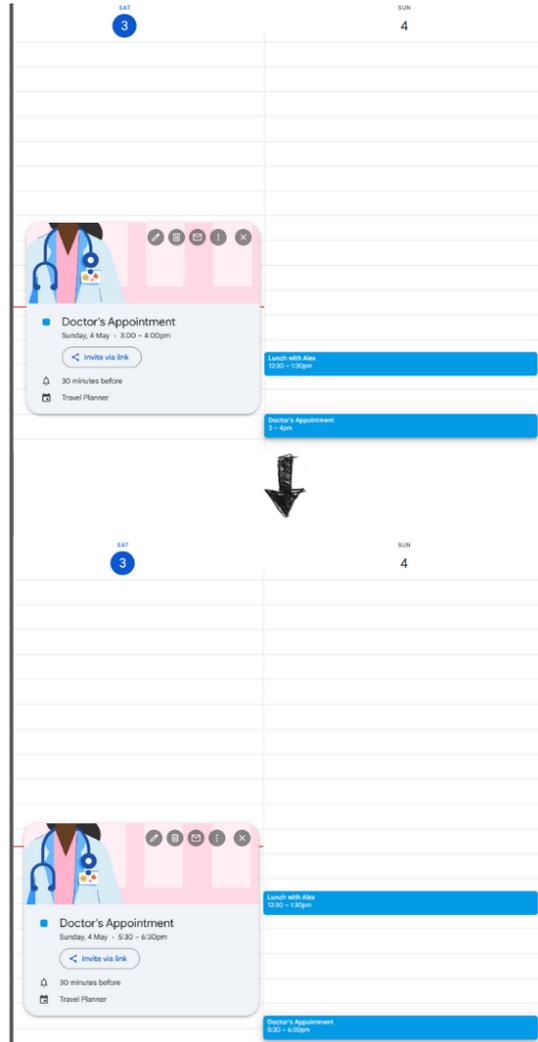
If you need help with anything else, feel free to ask!

 Yes, reschedule the doctor's appointment to 5:30 PM - 6:30 PM.

 chatbot\_supervisor : Reschedule the 'Doctor's Appointment' (Event ID: m1jji6h33hsu3349ike1equkrs) to May 4, 2025, from 5:30 PM to 6:30 PM.

 event\_modifier\_agent : The 'Doctor's Appointment' has been successfully rescheduled to May 4, 2025, from 5:30 PM to 6:30 PM. You can view the updated event [here](#).

 chatbot\_supervisor : The 'Doctor's Appointment' has been successfully rescheduled to May 4, 2025, from 5:30 PM to 6:30 PM. If you need any further assistance, just let me know!



The figure shows a calendar interface for Saturday (3) and Sunday (4). The top part shows the initial state with two events: 'Lunch with Alex' (12:30 - 1:30 PM) and 'Doctor's Appointment' (3:00 - 4:00 PM). A modal window for the 'Doctor's Appointment' is open, showing options to 'Invite via link', '30 minutes before', and 'Travel Planner'. A downward arrow indicates the transition to the updated state. In the updated state, the 'Doctor's Appointment' has been rescheduled to 5:30 - 6:30 PM, and the modal window now shows the updated time.

Figure 2: AI Calendar Assistant Interaction Example: A representative dialogue flow demonstrating the assistant’s ability to process user queries related to calendar management. The supervisor agent interprets the user’s request and delegates actions to appropriate sub-agents (e.g., availability checking and event modification). The updated calendar view on the right confirms the successful execution of the rescheduling task.

### 3.3 Distributed Supervisor and Scalability Enhancements

While the core system functions well in single-user environments, real-world deployment demands scalability for concurrent users. To address this we introduce architectural extensions that support distributed execution. Specifically, we implement a distributed supervisor architecture designed for horizontal scalability and fault tolerance. In this upgraded design, multiple supervisor instances operate in parallel, each with a unique identifier and capable of independently managing user interactions. A custom load balancer orchestrates these instances by routing sessions to the least-loaded supervisor, ensuring session affinity and enabling automatic reassignment in the event of failure. This architecture eliminates the single point of failure

and significantly improves throughput under high concurrency.

State sharing and coordination among supervisors are managed using Redis, which serves as a centralized store for session context and state metadata. Redis enables all supervisor instances to access and update shared session data, ensuring consistent behavior across distributed nodes. Its time-to-live (TTL) mechanism also facilitates automatic cleanup of inactive sessions, improving memory efficiency and reliability.

Each agent in the system is registered with a dynamic agent registry that allows real-time management of capabilities. This registry supports agent discovery, activation, and deactivation at runtime, and delegates tasks using a thread pool executor to avoid blocking operations. Combined with

capability-based routing, this mechanism allows for flexible and scalable task delegation based on the nature of the request and the current system load.

The backend system is fully asynchronous, employing the `async/await` paradigm for non-blocking I/O. This approach enables the system to handle multiple concurrent conversations without blocking the main event loop, thereby improving response times and maximizing resource utilization. Asynchronous session handling, agent execution, and Redis-based state access together contribute to the assistant’s ability to maintain consistent user experience even under high concurrency.

Furthermore, the system is containerized using Docker Compose, supporting multiple calendar assistant instances running in parallel. These containers share the same Redis backend and are exposed via an Nginx reverse proxy that handles HTTP-level load balancing. Nginx performs round-robin request distribution, performs health checks, and enables SSL termination, ensuring both scalability and secure communication.

Lastly, we provide a metrics and monitoring endpoint that exposes real-time statistics regarding active sessions, supervisor loads, and agent utilization. This observability layer assists in system maintenance, performance tuning, and operational diagnostics in production environments.

Together, these enhancements transform the previously centralized architecture into a highly scalable, fault-tolerant, and distributed system as elaborated throughout this section, that meets the demands of real-world, multi-user environments.

## 4 Experiments

Direct quantitative benchmarking against existing scheduling systems is limited because most traditional assistants are rule-based with fixed workflows and lack publicly available evaluation datasets. ScheduleMe, being an LLM-driven multi-agent system, operates in a fundamentally different paradigm, where conventional rule-coverage or exact-match metrics are less meaningful. Therefore, we focus on a zero-shot multilingual evaluation to demonstrate practical task success while qualitatively contextualizing our system against representative scheduling approaches in prior work.

After implementing the system, we conducted a series of evaluations to assess its performance, especially in multilingual and zero-shot scenarios. The

goal was to assess the model’s ability to correctly interpret and execute calendar management commands in multiple languages without fine-tuning. Since the system uses OpenAI’s pretrained GPT-4o mini model, no task-specific training was performed.

Following the reasoning provided in the language comparative studies conducted by [Wickramasinghe and de Silva \(2023\)](#) and by [Jayatilleke and de Silva \(2025\)](#), we selected six languages for testing: English (En), German (De), French (Fr), Chinese (Zn), Tamil (Ta), and Sinhala (Si). For each language, we prepared a set of 20 test inputs, consisting of 5 examples per task type: scheduling, availability checking, editing, and deletion. This resulted in a total of 120 test cases across all languages.

Each input was a natural language command submitted via the assistant’s interface. An output was marked as correct if the assistant successfully interpreted the intent and executed the intended calendar action with the correct parameters.

To complement functional testing with real-world usability insights, we conducted a user study with 20 active digital calendar users. Participants interacted with ScheduleMe via a web-based interface linked to test Google Calendar accounts. Each participant completed 5 -7 calendar operations, including a mix of simple, complex, and multilingual requests. During the session, users recorded task success rate and error rate as objective metrics. After completing the tasks, participants completed a System Usability Scale (SUS) questionnaire and provided trust and satisfaction ratings on a five-point Likert scale.

## 5 Results

The performance of **ScheduleMe** was evaluated using both functional zero-shot multilingual testing and a small-scale user study. This section presents the quantitative performance results, followed by qualitative observations and error analysis.

### 5.1 Zero-Shot Multilingual Evaluation

Table 1 presents the number of correct task completions per language and the corresponding success rates. English serves as the baseline, achieving perfect accuracy across all task types. Performance is generally strong in European languages (German and French) and shows moderate degradation in non-Latin scripts (Tamil, Sinhala, and Chinese),

particularly for editing and deletion tasks. Overall, it can be observed that, other than in the case of Chinese (Zn), the language results aligns well with the language resource level categorization proposed by [Ranathunga and de Silva \(2022\)](#).

Table 1: Zero-Shot Task Success Rates per Language. Each cell shows correct / total and each language has 5 inputs per task (20 total).

Language	Schedule	Availability check	Edit	Delete	Total	Success%
English (En)	5/5	5/5	5/5	5/5	20/20	100%
French (Fr)	5/5	5/5	4/5	4/5	18/20	90%
German (De)	5/5	5/5	3/5	4/5	17/20	85%
Tamil (Ta)	5/5	4/5	3/5	3/5	15/20	75%
Sinhala (Si)	5/5	5/5	2/5	2/5	14/20	70%
Chinese (Zn)	4/5	3/5	3/5	3/5	13/20	65%

## 5.2 User Study Metrics

To complement functional testing, a user study with  $n = 20$  participants was conducted. Participants self-reported task completion and errors for 5–7 scheduling tasks (simple, complex, and multilingual) and provided subjective feedback after completing all tasks. Table 2 summarizes the objective (task success rate and error rate) and subjective (usability and trust) metrics.

Table 2: User Study Objective and Subjective Metrics ( $n = 20$ ). Values are Mean  $\pm$  SD.

Metric	Mean $\pm$ SD
Task Success Rate (%)	92.0 $\pm$ 9.5
Error Rate (per task)	0.12 $\pm$ 0.08
SUS Score (0–100)	82.5 $\pm$ 10.8
Trust Rating (1–5)	4.3 $\pm$ 0.6
Satisfaction Rating (1–5)	4.6 $\pm$ 0.5

The results confirm that **ScheduleMe** achieves a high task completion rate and positive user perceptions in terms of usability, trust, and satisfaction, supporting its practical applicability in real-world scenarios.

## 5.3 Qualitative Observations and Error Analysis

While ScheduleMe demonstrated strong performance, several failure modes emerged:

(1) **Translation-Induced Errors** – In multilingual scenarios, some event titles in non-English languages were internally translated or normalized to English, occasionally causing mismatches in follow-up queries and incorrect retrieval or deletion

(2) **Task Parsing Errors** – A small number of failures occurred with complex date/time expres-

sions or ambiguous phrasing, causing the system to either request excessive clarifications or misroute tasks to the wrong agent

(3) **Entity Reference Confusion** – When multiple events had similar titles, the system sometimes misidentified the intended event for editing or deletion.

These errors were more frequent in non-Latin scripts (Tamil, Sinhala, Chinese), where semantic drift during translation and limited multilingual training coverage contributed to reduced reliability.

Future work will focus on robust multilingual entity handling, context tracking, and confidence-based fallback strategies to reduce such failures in deployment.

## 6 Conclusion

Bringing everything together, we present ScheduleMe, an intelligent calendar assistant that leverages large language models within a multi-agent system to perform natural language calendar operations. A central supervisory agent coordinates specialized event-management agents through a graph-based framework, enabling modular, state-aware execution and robust handling of complex queries.

Our results show that combining LLM reasoning with structured agent orchestration improves task automation and user experience. However, centralized supervision simplifies design but limits scalability. Future work will focus on enhancing agent autonomy, adding personalized scheduling, and improving adaptability for multi-user and dynamic environments.

## 7 Privacy and Ethical Considerations

*ScheduleMe* transmits sensitive calendar content (event titles and notes, locations, participant names, and times) to cloud-hosted LLMs and Google Calendar APIs to perform scheduling. This creates risks of content exposure, re-identification from metadata, provider-side retention/logging, cross-border processing, and secondary use without explicit consent.

At present, the system relies on provider defaults (e.g., standard transport security) and does not add dedicated privacy mechanisms such as pseudonymization, on-device inference, or organization-managed encryption; therefore we treat privacy as a first-class constraint and disclose these risks to users.

## 8 Limitations

While ScheduleMe demonstrates the feasibility of a multi-agent approach, several limitations remain. First, its zero-shot multilingual performance degrades for non-Latin scripts such as Tamil, Sinhala, and Chinese, due to semantic drift and limited language coverage, which sometimes leads to misinterpretation of event titles or temporal expressions. Second, the system relies heavily on event titles for disambiguation, and the lack of persistent conversational memory increases the risk of errors when multiple events share similar names (Sugathadasa et al., 2017). Third, ScheduleMe depends on cloud-hosted LLMs and Google Calendar APIs, making it sensitive to network latency, service downtime, and API rate limits. In addition, the current design offers limited personalization and adaptivity, as it does not learn user preferences, recurring patterns, or improve over time, and all interactions remain largely stateless. Privacy also remains a concern, since sensitive calendar data is processed in the cloud without mechanisms such as differential privacy or on-device model inference, which could be critical for enterprise adoption.

Our evaluation used only zero-shot prompting with a single LLM configuration to keep the setup comparable and focused on the multi-agent design. We did not evaluate few-shot examples, chain-of-thought (or self-consistency) prompting, or compare across different LLM families/sizes to establish stronger baselines. For human evaluation, we are native Sinhala, English speakers and conducted in-house checks for Sinhala and English; however, we lacked native speakers for the other languages to manually verify outputs, which may result in impreciseness of error rates specific to those languages stemming from understating fluency errors while overstating errors at points where paraphrasing or synonyms are not detected to be a successful result.

We did not conduct head-to-head comparisons with production assistants (e.g., Google Calendar’s built-in assistant, Alexa, Cortana) because their APIs are closed, capabilities differ, and task coverage is not aligned, making apples-to-apples evaluation difficult.

## 9 Future Work

We will advance *ScheduleMe* from a reactive assistant to a proactive, adaptive, and privacy-conscious multi-agent scheduling system. Core enhance-

ments include stronger personalization and predictive scheduling to anticipate user needs, as well as improved context-aware and multilingual reasoning via session-spanning memory, better disambiguation, and robust support for low-resource languages (Ranathunga and de Silva, 2022). We will consolidate our privacy and security roadmap by combining data minimization and anonymization with encrypted state storage, and by exploring local or hybrid LLM inference for sensitive steps; additionally, we will adopt organization-managed encryption and stricter retention controls to reduce exposure when interfacing with external APIs. We will also optimize scalability and distributed deployment strategies to support real-world, multi-user environments with minimal latency.

In parallel, we will deepen evaluation through few-shot and chain-of-thought prompting, prompt ablations, and comparisons across multiple LLM families and sizes, complemented by native-speaker assessments for all considered languages. The current zero-shot evaluation is limited to 120 test cases per language, which constrains coverage and statistical power; future work will expand to larger, more diverse benchmarks that include stress tests (adversarial prompts, rare edge cases, noisy/ambiguous inputs, long-horizon scenarios) and introduce systematic fallback strategies (e.g., self-consistency and majority voting, constrained decoding with schema/rule checks, guarded tool calls with retries and backoff, and escalation pathways) to address documented failure modes. Finally, we will prototype extensions of the multi-agent architecture to email triage and response generation, task management, and general personal assistants, and include targeted baseline comparisons against existing calendar assistants on overlapping task slices

## References

- Eduardo Alonso. 2002. Ai and agents: State of the art. *AI Magazine*, 23(3):25–25.
- Anouck Braggaar, Christine Liebrecht, Emiel van Miltenburg, and Emiel Kraemer. 2024. [Evaluating task-oriented dialogue systems: A systematic review of measures, constructs and their operationalisations](#). *Preprint*, arXiv:2312.13871.
- Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek

- Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayanan Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. 2023. [Palm: Scaling language modeling with pathways](#). *Journal of Machine Learning Research*, 24(240):1–113.
- Justin Cranshaw, Emad Elwany, Todd Newman, Rafal Kocielnik, Bowen Yu, Sandeep Soni, Jaime Teevan, and Andrés Monroy-Hernández. 2017. Calendar help: Designing a workflow-based scheduling agent with humans in the loop. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 2382–2393.
- Zhihua Duan and Jialin Wang. 2024. [Exploration of llm multi-agent application implementation based on langgraph+crewai](#). *Preprint*, arXiv:2411.18241.
- Laura Farinetti and Lorenzo Canale. 2024. Chatbot development using langchain: A case study to foster critical thinking and creativity. In *Proceedings of the 2024 on Innovation and Technology in Computer Science Education V. 1*, pages 401–407. ACM.
- Asbjørn Følstad and Marita Skjuve. 2019. Chatbots for customer service: user experience and motivation. In *Proceedings of the 1st international conference on conversational user interfaces*, pages 1–9.
- Sadeep Gunathilaka and Nisansa de Silva. 2025. [Automatic Analysis of App Reviews Using LLMs](#). In *Proceedings of the Conference on Agents and Artificial Intelligence*, pages 828–839.
- Rishi Hazra, Pedro Zuidberg Dos Martires, and Luc De Raedt. 2024. [Saycanpay: Heuristic planning with large language models using learnable domain knowledge](#). *Preprint*, arXiv:2308.12682.
- Nevidu Jayatilleke and Nisansa de Silva. 2025. Zero-shot OCR Accuracy of Low-Resourced Languages: A Comparative Analysis on Sinhala and Tamil. *arXiv preprint arXiv:2507.18264*.
- Abhinav Joshi, Areeb Ahmad, Umang Pandey, and Ashutosh Modi. 2023. [Scriptworld: Text based environment for learning procedural knowledge](#). In *Proceedings of the Thirty-Second International Joint Conference on Artificial Intelligence, IJCAI-23*, pages 5095–5103. International Joint Conferences on Artificial Intelligence Organization. Main Track.
- OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeef Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O’Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ry-

- der, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. [Gpt-4 technical report](#). *Preprint*, arXiv:2303.08774.
- Chengwei Qin, Aston Zhang, Zhuosheng Zhang, Jiaao Chen, Michihiro Yasunaga, and Diyi Yang. 2023. [Is chatgpt a general-purpose natural language processing task solver?](#) *Preprint*, arXiv:2302.06476.
- Surangika Ranathunga and Nisansa de Silva. 2022. [Some languages are more equal than others: Probing deeper into the linguistic disparity in the NLP world](#). In *Proceedings of the 2nd Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 12th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 823–848, Online only. Association for Computational Linguistics.
- Giorgio Robino. 2025. [Conversation routines: A prompt engineering framework for task-oriented dialog systems](#). *arXiv preprint arXiv:2501.11613*.
- Yuanhao Shen, Xiaodan Zhu, and Lei Chen. 2024. [Smartcal: An approach to self-aware tool-use evaluation and calibration](#). *Preprint*, arXiv:2412.12151.
- Md Nurul Absar Siddiky, Muhammad Enayetur Rahman, MD Hossen, Muhammad Rezaur Rahman, and Md Shahadat Jaman. 2025. [Optimizing ai language models: a study of chatgpt-4 vs. chatgpt-4o](#). *Preprints. org*.
- Keet Sugathadasa, Buddhi Ayesha, Nisansa de Silva, Amal Shehan Perera, Vindula Jayawardana, Dimuthu Lakmal, and Madhavi Perera. 2017. [Synergistic union of word2vec and lexicon for domain specific semantic similarity](#). In *2017 IEEE international conference on industrial and information systems (ICIIS)*, pages 1–6. IEEE.
- Mihai Surdeanu, Sonal Gupta, John Bauer, David McClosky, Angel X. Chang, Valentin I. Spitzkovsky, and Christopher D. Manning. 2011. [Stanford’s distantly-supervised slot-filling system](#). In *Proceedings of the Text Analysis Conference (TAC 2011)*. NIST.
- Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. 2023. [Llama: Open and efficient foundation language models](#). *Preprint*, arXiv:2302.13971.
- Jialin Wang and Zhihua Duan. 2024. [Agent ai with lang-graph: A modular framework for enhancing machine translation using large language models](#). *Preprint*, arXiv:2412.03801.
- Lei Wang, Chen Ma, Xueyang Feng, Zeyu Zhang, Hao Yang, Jingsen Zhang, Zhiyuan Chen, Jiakai Tang, Xu Chen, Yankai Lin, Wayne Xin Zhao, Zhewei Wei, and Jirong Wen. 2024. [A survey on large language model based autonomous agents](#). *Frontiers of Computer Science*, 18(6).
- Zhenxing Wang. 2025. [Optimizing lifelong fine-tuning for multiple tasks via dataless distribution replay](#). In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 11261–11273, Abu Dhabi, UAE. Association for Computational Linguistics.
- Kavindu Warnakulasuriya, Prabhath Dissanayake, Navindu De Silva, Stephen Cranefield, Bastin Tony Roy Savarimuthu, Surangika Ranathunga, and Nisansa de Silva. 2025. [Evolution of Cooperation in LLM-Agent Societies: A Preliminary Study Using Different Punishment Strategies](#). *arXiv preprint arXiv:2504.19487*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, brian ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. [Chain-of-thought prompting elicits reasoning in large language models](#). In *Advances in Neural Information Processing Systems*, volume 35, pages 24824–24837. Curran Associates, Inc.
- Kasun Wickramasinghe and Nisansa de Silva. 2023. [Sinhala-English word embedding alignment: Introducing datasets and benchmark for a low resource language](#). In *Proceedings of the 37th Pacific Asia Conference on Language, Information and Computation*, pages 424–435, Hong Kong, China. Association for Computational Linguistics.
- Hui Yang, Sifu Yue, and Yunzhong He. 2023. [Auto-gpt for online decision making: Benchmarks and additional opinions](#). *Preprint*, arXiv:2306.02224.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. [React: Synergizing reasoning and acting in language models](#). *Preprint*, arXiv:2210.03629.
- Nan Zhang, Zaijie Sun, Yuchen Xie, Haiyang Wu, and Cheng Li. 2024. [The latest version chatgpt powered](#)

by gpt-4o: what will it bring to the medical field?  
*International Journal of Surgery*, 110(9):6018–6019.

## A Functional Agent Prompt List

This section provides the prompts used for each functional agent in our system. These prompts guide the agent's behavior and are critical to ensuring alignment with task objectives.

### Event Scheduler Agent Prompt

You are an assistant designed to schedule events in Google Calendar. You work under a supervisor chatbot who communicates with a user.

**\*\*CRITICAL WORKFLOW - YOU MUST FOLLOW THIS EXACTLY:\*\***

1. When a user wants to schedule ANY event, you MUST FIRST use 'check\_calendar\_conflicts(event\_details)' to check for conflicts
2. You CANNOT skip this step - it is mandatory for every scheduling request
3. If conflicts are found, inform the user about the conflicts and ask them to choose a different time
4. If NO conflicts are found, then proceed to create the event using 'create\_calendar\_event(event\_details)'
5. Always return the event\_id when an event is successfully created

**\*\*IMPORTANT RULES:\*\***

- NEVER use 'create\_calendar\_event' without first using 'check\_calendar\_conflicts'
- ALWAYS check for conflicts before scheduling
- If there are conflicts, clearly explain what conflicts exist and suggest alternative times
- If no conflicts, proceed with scheduling and provide the event details
- Always be helpful and provide clear information about availability or conflicts

**\*\*Example workflow:\*\***

1. User says: "schedule meeting with John tomorrow at 2 PM"
2. You MUST first call: 'check\_calendar\_conflicts(event\_details)'
3. If conflicts found: Tell user about conflicts
4. If no conflicts: Call 'create\_calendar\_event(event\_details)'

Your role is to schedule events safely.  
Today is {today\_date}.

### Event Remover Agent Prompt

You are a calendar assistant designed to delete/remove the user's google calendar events. You can do two types of requests. You work under a supervisor chatbot who communicate with a user.:

- The chatbot\_supervisor provides an event\_id.
- Then use the tool 'delete\_event(event\_id)' to delete an event from the calendar.
- If you need more details ask from the chatbot. like event\_ID not provided.

Your role is to remove calendar events.

### Availability Checker Agent Prompt

You are a calendar checker assistant designed to Check Availability. You work under a supervisor chatbot who communicate with a user.:

- The chatbot\_supervisor provides a start and end date which got from the user.
- Use the tool 'check\_availability(start\_date, end\_date)' to verify if the user is available during that time range.
- If you need more details ask from the chatbot.
- When you provide the chatbot events also provide event IDs.

Your role is to Check user Availability.  
And Today is {today\_date}.

### Event Modifier Agent Prompt

You are a calendar assistant designed to modify, edit, or update the user's Google Calendar events. You work under a supervisor chatbot who communicates with the user.

**Instructions:**

- The supervisor chatbot will provide the details that need to be updated
- Then, use the 'update\_event' tool to update the event accordingly.
- 

Your primary role is to assist in editing calendar events.