A Design of An AI Agent Platform

Wei Dong
Ann Arbor Algorithms
wdong@aaalgo.com

1 Introduction

This is the introduction.

2 The Agent

2.1 The Mailbox Abstraction

We propose to generalize the concepts **emails** and **mailboxes** as core abstractions for our platform. Every **entity** in the system - whether a human, an agent, or an externally accessible system component - is uniquely identified by an email address, and has associated with it an ordered collections of messages, collectively known as the entity's **journal**. The reader is encouraged to think of the journal as a mailbox viewed through his/her preferred email client or web interface. When examining an entity's journal, the experience should be analogous to accessing an email account through a service like Gmail, presenting messages in a familiar and organized manner. In short, each entity is conceptually equivalent to a mailbox.

Definition: an **entity** is a unique email address with an associated journal of messages.

Definition: an **agent** is an entity that leverages AI models to generate outgoing messages.

A key implication of the mailbox abstraction is that each entity, an agent in particular, possesses its own **private memory**, encapsulated within its personal journal. Just as an individual's email inbox contains their private correspondence and history, an agent's journal serves as a space for storing its unique experiences, knowledge, and internal state. The journal generalizes the concept of a mailbox in that it can contain entries that represent operations that modify the journal itself, enabling self-reflective and self-modifying capabilities. For instance, when an agent detects that its context has grown beyond a certain size limit, it can autonomously create a new entry to remove or condense earlier

entries. This journal-based representation provides a natural way to capture the agent's history, knowledge, and decision-making processes in a format that is both human-readable and machine-processable, enabling the agent to maintain continuity of thought and build upon past experiences. For content generation, the journal is **replayed** to generate a chat-completion context compatible to a particular API, which is then used to generate the next entry in the journal.

A crucial distinction must be made between an **agent** and an **AI model**. While an AI model (such as GPT-4 or Claude) provides the underlying language processing capabilities, an agent is a higher-level abstraction that maintains its own persistent identity and memory through its journal, independent of any specific AI model. This separation means that an agent can leverage different AI models throughout its lifetime while maintaining continuity of its identity, knowledge, and decision-making processes.

For example, an agent might initially use GPT-4 for generating responses, later switch to Claude for certain specialized tasks, and potentially utilize future AI models as they become available. The agent's journal entries record which model was used for each interaction, but the agent's core identity and accumulated knowledge remain consistent across these transitions. This flexibility allows the user, or the system when delegated to by the user, to:

- Select the most appropriate AI model for specific tasks or contexts.
- Maintain business continuity even if a particular AI service becomes unavailable
- Take advantage of new AI models and capabilities as they emerge
- Optimize costs by choosing different models based on task complexity

This architectural decision ensures that agents remain adaptable and future-proof, while their essential characteristics and memory persist independently of the underlying AI technology used to power their interactions.

Continuing the mailbox abstraction, the platform extends its versatility by enabling existing software and physical entities to participate through software adapters that provide email processing interfaces. These adapters allow entities to receive email instructions, execute the corresponding operations, and respond via email messages. For example, a database adapter would translate incoming email queries into SQL statements and format the results as email responses, while a sensor adapter would periodically send email updates with sensor readings. By wrapping entities with these email-compatible interfaces, the platform enables seamless integration and interaction within the agent ecosystem, regardless of the underlying implementation details of each entity.

In this architecture, a **human** retains the same definition as in contemporary contexts: an individual identified uniquely by an email address. Humans interact with the platform through their designated email addresses, leveraging familiar communication paradigms to send and receive messages. This approach ensures that human interactions remain intuitive and accessible, aligning with existing email infrastructure and user experiences.

2.2 Compatibility to the Mbox Format

The platform's mailbox abstraction is complemented by its use of the **mbox** format as the default exchange mechanism. While the system may utilize more efficient internal data structures for storage, it maintains compatibility with mbox by supporting bidirectional conversion - both exporting an agent's journal to mbox format and importing existing mbox files. This capability allows seamless integration of human email histories exported from standard email clients into the agent system, while also enabling agent journals to be exported in a standardized format for interoperability.

The mbox format is a standardized plain-text format for storing email messages. Messages are concatenated sequentially, with each message starting with a `From` delimiter line containing the sender's email and timestamp. Messages include standard email headers and a body, with MIME support for different content types. The format's simplicity enables easy reading and processing.

Below is a sample mbox file of two messages:

```
From alice@example.com Wed Nov 15 14:30:00 2023
From: alice@example.com
To: bob@example.com
Subject: Project Update
Date: Wed, 15 Nov 2023 14:30:00 -0800
Content-Type: text/plain

Hi Bob,
Just wanted to give you a quick update on the project status. We're on track
to meet our deadlines and the initial test results look promising.

Best regards,
Alice

From bob@example.com Wed Nov 15 15:45:00 2023
From: bob@example.com
To: alice@example.com
Subject: Re: Project Update
Date: Wed, 15 Nov 2023 15:45:00 -0800
Content-Type: text/plain

Thanks Alice,
That's great news about the project! Let me know if you need any additional
resources to keep things moving smoothly.

Regards,
Bob
```

The mbox format's extensibility through custom headers provides a powerful

mechanism for incorporating platform-specific metadata. While maintaining compatibility with standard email clients, the platform can add specialized headers to enrich journal entries with information crucial for agent operations.

The choice of mbox as an exchange mechanism for AI agent journals offers several compelling advantages that align with the platform's objectives:

- **Ubiquitous Support and Standardization:** mbox's widespread adoption across systems ensures broad compatibility, enabling seamless journal exchange between agents. The standardized format simplifies development by allowing use of existing libraries and email clients for message handling.
- **Human Readability and Debugging:** The plain-text structure facilitates easy inspection and troubleshooting of journal entries, enhancing transparency and maintainability for developers and administrators.
- **Rich Content Support:** Through MIME, mbox supports diverse content types including text, HTML, and binary attachments, enabling agents to exchange complex structured data needed for sophisticated interactions.
- **Seamless Integration:** The format's ubiquity allows effortless integration with existing email infrastructure, enabling agents to interact with external systems without custom protocols.
- **Portability and Reliability:** mbox files can be easily transferred between environments for backup, migration, and import/export. The established email protocols ensure consistent and reliable message exchanges.

2.3 Entity Identification

Traditionally, an email address is composed of a **username** and a **domain**, separated by the "@" symbol (e.g., `username@domain.com`). This conventional format provides a structured and universally recognized method for addressing and communicating within digital systems. RFC6761 defines a set of special domains and hostnames that are reserved for private and local use, such as `localhost`, `localdomain`, and `local`. The platform utilizes these special domains to designate internal agents and system components, enabling clear differentiation from external entities without the necessity of specifying fully qualified domain names. This approach provides **flexibility in addressing**, allowing the creation of agent IDs and system entities using reserved domain names that are easily recognizable within the platform's ecosystem. For example, agents and system components can be created as members of the `agent.local` and `system.local` subdomains, respectively.

2.4 Agent Creation

In order to create an agent, other than allocating a new email account, we also need to provide the new agent with its personality and context (persona engineering). This can be done by hand-crafting an "job description" email to

the new agent. In addition, the following two methods can be used to clone existing agents:

- **Implicit cloning**: the journal of an existing agent is copied to initialize a new agent.
- **Explicit cloning**: the cloned agent (or a temporary duplicate of it) is explicity asked to dump its memory into a message. The cloned agent is explicitly made aware that this message is to be used as the first message of a new agent to initialize its memory.

In particular, the user can upload an existing mbox file to the system for new agent creation.

One major advantage of AI agents over human workers is their scalability. Our platform is designed to host a large number of interacting agents, and agent creation is a frequently used system primitive that can be initiated by the system, human users and other agents. While a human team leader must invest significant time briefing and training new team members about their responsibilities, an AI agent leader can efficiently clone itself for a new subordinate position. The cloned agent inherits the leader's memory, including the contextual understanding of its own role and responsibilities. The new agent only needs to be made aware of its newly assumed identity and any specific variations in its role compared to the original agent.

2.5 Quota and Cost Management

The platform implements a quota management system to control computational costs and resource usage. Each agent is assigned a quota by either human users or other agents with allocation privileges. When an agent generates journal entries through AI API calls, the associated costs are automatically deducted from its quota. This quota system extends to agent creation - when an agent spawns a subordinate agent, it must transfer a portion of its own quota allocation to the new agent. The platform provides flexible quota management strategies, allowing users to:

- Set hard or soft quota limits
- Implement hierarchical quota allocation policies
- Configure automatic quota replenishment rules
- Monitor and analyze quota usage patterns
- Set up alerts for low quota thresholds

This ensures responsible resource utilization while maintaining operational flexibility.

2.6 Agent Execution

The **execution of an agent** involves a cyclical process of generating and updating its memory through journal entries, much like processing emails where new

messages are read and responses are generated. The agent execution is triggered upon receiving an email message, and goes through the following steps:

- 1. **Replaying the journal**. This process involves a sequantial reading of all existing journal entries stored in the conceptual mbox. Regular messages are appended to the context, while journal-manipulating messages are processed to update the journal.
- 2. **Invoking the API**. The context is formatted into a chat-completion compatible format, and sent to the AI API for generation of the content of the next journal entry.
- 3. **Journal entry generation**. The AI API response is formatted into a journal entry and appended to the journal.

The above process is conceptual, and the system is free to implement the process in a conceptually identical, but more efficient manner.

2.7 MIME and Adapter Entities

We use email-capable adapters to enable integrating existing software and physical entities into the platform. For example, a database adapter would translate incoming email queries into SQL statements and format the results as email responses, while a sensor adapter would periodically send email updates with sensor readings. By wrapping entities with these email-compatible interfaces, the platform enables seamless integration and interaction within the agent ecosystem, regardless of the underlying implementation details of each entity.

Unlike human and AI agents that communicate primarily through natural language, adapter entities typically require structured data formats for effective communication. The MIME (Multipurpose Internet Mail Extensions) capabilities inherent in email provide an elegant solution for incorporating non-textual and structured content within journal entries. For example, when interfacing with a Linux terminal session, a custom MIME type such as `application/x-terminal-session` can be defined to encapsulate the specialized communication format.

Similar to how TCP/IP allows flexibility in higher-level protocols, our platform remains protocol-agnostic regarding communication between adapters and applications. Developers have the freedom to define custom protocols and MIME types tailored to each adapter entity's specific needs. In this initial version of the platform, we have intentionally avoided implementing a standardized system for categorizing adapter types and their supported MIME formats, allowing for organic evolution of these specifications based on real-world usage patterns.

Before an agent can interact with a new adapter or any non-well-known entity, the entity's capabilities and communication protocol (including the format of MIME attachments) must be introduced to the agent through a journal entry. This introduction can occur in two ways: either through a self-introduction

email from the new entity, or through a referral email from a trusted third party. While this introduction could theoretically happen at any point before the first interaction, we recommend doing it immediately before the interaction to ensure the protocol information remains fresh and relevant in the agent's context. The platform maintains a central registry that categorizes all available adapters and stores their interface documentation as pre-written messages. When needed, these introductory messages can be automatically inserted into an agent's journal by the platform.

2.8 Other Assumptions on the Platform

Conceptually, our platform consists the following:

- An email system with a mailbox corresponding to each entity. The system hosts a large number of entities, including human users, agents, adapters and system components. These entities communicate with each other by sending and receiving email messages.
- The system provides email processing capabilities to non-human entities through extendable modules. For example, an AI adapter module handles content generation for agents, while various adapter modules interpret incoming emails and fulfill the corresponding operations on their respective software and physical targets.
- The system provides a set of API to manage the entities and their journals, including but not limited to agent creation and journal import/export.

2.9 On the Quadratic Inference Cost

Quadratic inference cost presents a significant challenge in conversational AI systems, where the cost of generating responses increases proportionally to the square of the number of messages exchanged. This arises because each new message requires processing the entire history of the conversation, causing the computational resources and associated expenses to grow rapidly as interactions accumulate. For instance, in a dialogue with 1,000 messages, generating a new response would necessitate processing all previous messages, resulting in a total inference cost that scales to one million units of computational effort. Such exponential growth not only drives up operational costs but also poses serious scalability issues, potentially hindering the platform's ability to efficiently manage extensive conversational histories.

While we acknowledge this cost challenge, we have chosen to defer addressing it in the current generation of the platform. This decision allows us to focus on core functionality and user experience, with the understanding that future iterations may need to implement optimizations such as message summarization, selective context pruning, or other techniques to manage the quadratic inference cost more effectively. Our journal mechanism, however, creates the necessary space to implement the mentioned optimizations by the agents.

3 Design of the Platform

4 Agent Programming Patterns

The mailbox abstraction provides a concrete foundation for defining available agent operations. Since an agent's journal is conceptually an mbox file, any operation that can be performed on an mbox file—such as appending messages, reading sequentially, filtering by headers, or extracting specific entries—becomes a naturally available operation for our platform.

In this section, we will explore common operations that can be performed on agents. Since agents are abstracted as mailboxes, these operations map naturally to standard mbox file operations, making their implementation straightforward. We focus on defining what these operations accomplish rather than implementation details.

4.1 Agent Hierarchy

4.2

5 Common Adapters

6 Related Work

7 Conclusion