ChatGPT

Tongkeeper AI Agent – Feature Plan & Technical Stack

Introduction

Tongkeeper is a **generalist AI assistant** designed for personal and family use, built on the open-source Suna AI agent framework. It will provide a **chat-style interface** (similar to ChatGPT or Perplexity) for natural conversations and assistance. Tongkeeper's design emphasizes an intuitive UX (inspired by iOS design language) and robust capabilities such as scheduling, reminders, collaborative family chat, and task assistance. The system will initially deploy on the cloud (via Replit Agents) and use open-source Large Language Models (LLMs) – notably **DeepSeek LLM** as the primary model [1] – with a plan for future expansion to other models and modalities (like voice). Below, we outline the key product features (user-facing vs. system-level) and a recommended technical stack to achieve these goals.

User-Facing Features

- **Personalized Schedule Management & Reminders:** Users can create and manage personal events, deadlines, and reminders through natural language. Tongkeeper will understand commands like "Remind me to call Mom next Friday at 6pm" and schedule those events. The assistant can provide proactive alerts or daily agenda summaries. (Notably, even ChatGPT is moving toward this kind of functionality, adding a "Tasks" feature for scheduling reminders [2], underscoring the importance of this feature.) The UI will include a simple calendar or list view for upcoming reminders and send notifications when tasks are due.
- **Homework and Task Assistance:** Tongkeeper serves as a smart tutor and helper for household tasks or schoolwork. Users (including children) can ask questions about homework problems, get step-by-step explanations, or request help researching a topic. The assistant can break down complex problems, suggest resources, or even check work. For general tasks, it can provide to-do checklists or guide users through processes (e.g. **"Help me bake a cake – what steps do I follow?"**). Safeguards will be in place to encourage learning (e.g. providing hints rather than just answers for homework).
- **Natural Language Chatting:** At its core, Tongkeeper offers an **intuitive chat interface** for any kind of Q&A or conversation. Users can converse with the agent as they would with ChatGPT – asking for advice, information, or just casual chat. The interface will display user and assistant messages in a familiar chat bubble format, with a clean, minimalistic aesthetic (inspired by iMessage and ChatGPT UI for familiarity). Initially this is text-based, but designed such that voice input/output can be added later. The chat supports context awareness (the agent remembers earlier messages in the thread) and can reference the user's profile or past conversations for personalization.
- **Family Room Chat (Multi-User Group Chat):** Tongkeeper provides a shared **family chat room** where multiple family members can interact with the AI (and each other) together. One user (the host) can invite others via email to join the family chat space. All participants see the ongoing conversation and the AI's responses, enabling a collaborative experience – for example, planning a family trip together with Tongkeeper's help, or having a group Q&A session. The chat UI will indicate

different users (by their chosen profile names/avatars) and support real-time updates so everyone's messages and the AI responses appear instantly. This feature promotes inclusivity, letting parents and kids alike engage with the agent in a supervised environment.

- **Trip Planning Assistant:** Tongkeeper can serve as a virtual travel planner for the family. Users might say, *"Plan a 7-day vacation for our family to Japan next summer"*, and the agent will interactively suggest itineraries, attractions, and logistics. It can recommend destinations, generate day-by-day activity plans, check weather forecasts, and even draft packing checklists. Integration with external APIs (for weather, maps, or flight information) can enhance accuracy (for example, checking real-time weather to tailor outdoor vs. indoor activities [3] [4]). The planning results can be shared in the family chat, edited collaboratively, and saved as a document or itinerary within the app.
- **Long-Term Memory & Personalized Profiles:** Each user will have a **profile with long-term memory**. Users can set their name (or nickname) and share personal preferences (e.g. food dislikes, favorite subjects, birthdays) with Tongkeeper. The agent will remember these details over time – for instance, recalling a child's favorite dinosaur or a parent's coffee preference – to personalize interactions. This profile memory means if a user returns after days or weeks, Tongkeeper retains context (within privacy limits) of prior conversations and preferences. The system might greet users by name and can use stored info to enrich responses (e.g., reminding *"Your anniversary is next week, shall I set a reminder?"*). From a UI perspective, users can review and edit their profile data and see some history of key facts the AI has learned about them. This feature increases the feeling of a persistent, personalized companion rather than a stateless chatbot.

System-Level Features

- **LLM-Powered Natural Language Understanding:** At its core, Tongkeeper's intelligence comes from a Large Language Model. We will integrate **DeepSeek LLM** as the primary brain of the agent, leveraging its 67B-parameter capability for strong reasoning, coding, and general knowledge [1]. The system uses the LLM for understanding user inputs and generating appropriate responses in a conversational manner. All user queries across features (scheduling, homework help, etc.) are interpreted by this model. The integration is designed via an abstraction layer (as Suna does with LiteLLM) so that different model providers can be swapped in or combined – ensuring future flexibility to use models like Mistral or LLaMA as they become available or superior.
- **Dialog Management & Multi-User Context:** The backend will manage conversation **threads** and context for both one-on-one chats and multi-user family chats. It maintains who said what, and in multi-user sessions, associates messages with the correct user profile. The agent is capable of distinguishing users in the same chat and tailoring responses appropriately (e.g., using a child's name when answering their question). A thread management system ensures each conversation (or DM vs family room) has its own context state. This may involve tagging the speaker in the prompt so the AI knows the difference between users. System prompts might be used to enforce an AI persona appropriate for family use (e.g. a friendly, helpful tone, and avoiding inappropriate content).
- **Long-Term Memory Storage & Retrieval:** Tongkeeper implements a **memory architecture** to provide continuity and personalization. Key information from conversations and user profiles is persisted in a database and a vector store for semantic recall. For example, when a user shares a new piece of info ("my pet's name is Spot"), the system will store that as a memory. In future interactions, the backend can retrieve relevant facts and feed them into the LLM prompt to inform responses (so the AI can recall "Spot" when needed). We will use embeddings to allow semantic search in past dialogues (to find relevant context even if exact words differ). This long-term memory system ensures the AI **"remembers"** past interactions beyond the immediate context window. It also

includes conversation history saving, so users can scroll up to see previous chats, and the agent can summarize or draw from older messages when appropriate.

- **Personal Data and Profile Management:** The system includes robust **user management** under the hood. Each user's account (and family group membership) is stored securely. Authentication (likely email/password or magic link, possibly leveraging Supabase Auth or similar) ensures only invited users access a family room. The backend enforces access control (e.g. only family members can see that family's chat and data). Profile data (name, preferences, etc.) is stored in a structured form so it can be queried by the agent when constructing replies. Privacy controls are considered – e.g. users can mark certain profile fields as private (not to be used without consent) or wipe their data if needed. This subsystem handles invite links via email (generating a secure token that allows a new user to join a specific family chat upon sign-up).
- **Scheduling & Reminder Engine:** Underneath the user-facing scheduling feature is a **reminder service** that persists events and triggers notifications. This consists of storing events (with time, description, user, recurrence info) in the database and a background scheduler process that scans for due reminders. When a reminder time is reached, the system can notify the user – initially by generating a chat message from the agent (e.g. "*Reminder:* call Mom now") and later potentially via email or mobile push. The scheduling system may integrate with external calendars (for example, an OAuth integration to Google Calendar in the future) to sync events, but initially an internal calendar store suffices. The agent can also proactively mention upcoming items (e.g., each morning it could offer a summary of the day's schedule if asked). This requires the backend to support time-based job scheduling (which could be done via cron-like jobs or using a service like Supabase Edge Functions or a lightweight task queue).
- **Tool Integration Layer:** To fulfill complex requests, Tongkeeper will have a **tool use** subsystem. This allows the agent to perform actions like browsing the web, running computations, or calling external APIs to augment its LLM responses. Suna's architecture already includes a rich toolkit (browser automation, file management, web crawling, code execution, etc. [5]), and Tongkeeper can adopt a subset relevant to personal use. For instance, for trip planning, the agent might call a weather API or use a browser tool to look up attraction info. For homework help, it might use a math solver tool or a Python sandbox to run a calculation. Each tool is executed in a secure **sandbox environment** (Suna uses isolated Docker containers [6] for safety, ensuring code or web automation can't harm the host system). The agent's reasoning module will determine when a tool is needed and orchestrate the sequence of actions – e.g., search the web, then summarize results. This makes Tongkeeper more **action-oriented**, not just a text generator, capable of "acting" on the user's behalf much like Suna (which can browse, run code, etc. driven by conversational prompts [7] [8]).
- **Real-Time Communication & Notifications:** In support of the family chat and reminders, Tongkeeper's system includes **real-time messaging** capabilities. When one user sends a message or the AI posts a reply, all other clients in that chat should receive it instantly. To achieve this, the backend will use technologies like WebSockets or a publish/subscribe mechanism. (In Suna's design, a combination of Supabase Realtime and Redis is used – Supabase/Postgres for persistence and **Redis for real-time pub-sub and caching** [9].) Tongkeeper will adopt a similar approach: for example, broadcasting new chat messages to all web clients listening on a channel for that family room. This ensures a smooth, live chat experience without needing manual refresh. Additionally, this system can handle sending out email invites and possibly email/SMS notifications for certain events (like a reminder trigger if user is offline).
- **Scalability & Cloud Deployment:** The system-level design ensures that Tongkeeper can scale and potentially transition to hybrid architecture. Initially, it will be **cloud-hosted** (on Replit's platform for ease of deployment). The backend will be stateless aside from the database, enabling horizontal scaling if needed (multiple instances connecting to the same DB and cache). We anticipate using

containerization (Docker) for reproducible environments, aligned with Suna's container-based agent execution [6]. The design also keeps in mind a possible **hybrid model** later – e.g. running the core LLM inference on local family hardware or an edge device for privacy, while still using cloud for heavy tasks or cross-device sync. System-level modularity (separating the interface, logic, and data layers) will allow components to be moved or replicated across cloud and edge as needed in the future.

Technical Stack Recommendations

To implement the above features, we propose the following **technology stack and architecture** (leveraging the Suna AI agent's proven design):

Table 1: Tongkeeper Architecture – Components & Recommended Tech Stack

Layer / Component	Technology & Description
Large Language Model (LLM)	**DeepSeek LLM** (open-source 67B model) for the core conversational AI [1]. Integrated via an abstraction layer (e.g. Suna's **LiteLLM** or LangChain) so the agent can switch between LLM backends. This allows future use of other models (e.g. Mistral 7B, LLaMA2, or Google's Gemini) without rewriting core logic. Initially, DeepSeek can be accessed via a hosted API or a local inference server, given its size. The LLM handles all natural language understanding and generation tasks for Tongkeeper.
Backend Framework	**Python FastAPI** for the backend service (as used in Suna [10]). FastAPI provides an async web API to handle chat requests, user auth, scheduling events, etc. The backend implements endpoints for the chat UI (send/receive messages), user management, and integration with tools/APIs. FastAPI is lightweight yet powerful, and aligns with the Python ecosystem for AI (allowing integration of LangChain, numpy, etc.). If deploying on Replit, we can run FastAPI within Replit's environment (possibly behind a Flask-compatible interface or using Uvicorn ASGI server). FastAPI's asynchronous design helps when waiting on external calls (LLM API, web requests).
Frontend Framework	**Next.js (React)** for a responsive web UI [11]. This will deliver a **single-page app** that feels like a chat application, with components for the message list, input box, and maybe side panels for features like schedule or profile. Next.js allows server-side rendering if needed (for SEO or performance) and can easily be deployed (possibly on Vercel or Replit if supported). The UI will be designed following iOS guidelines – using clear typography, spacing, and familiar controls. For example, use of **Apple-like** UI elements (rounded text input, subtle shadows) to make it immediately approachable for iPhone/iPad users. We will include message bubbles (user messages on right, AI on left), and support rich content in messages (links, possibly images or cards for events). Frontend state management (via React context or Redux) will handle the multi-user chat sessions and personal data.

Layer / Component	Technology & Description
Real-Time Communication	**WebSockets or Supabase Realtime** for live updates. We can use FastAPI's WebSocket support (or Socket.IO) to push new chat messages to clients. Alternatively, leveraging **Supabase Realtime** (built on Postgres LISTEN/NOTIFY) allows clients to subscribe to database changes (e.g., new message rows) and update the UI in real-time. For simplicity, using WebSockets directly in the backend might be ideal – the server publishes any incoming message or AI response to a specific channel (room) so all clients in that room get it instantly. This real-time layer is crucial for the family chat feature. Additionally, **Redis** will be used under the hood as needed: for pub/sub to coordinate between multiple backend instances and for caching frequently accessed data (e.g., recent conversation context) [9] . Redis helps maintain snappy performance and synchronicity in a multi-user scenario.
Database & Persistence	**PostgreSQL (relational DB)** as the primary data store. We recommend using **Supabase** (a hosted Postgres solution with an API layer) as Suna does [12] , since it provides built-in auth and storage utilities. All structured data – user accounts, profiles, chat threads, messages, events, etc. – will live in Postgres tables. Supabase also offers file storage (useful if we allow the agent to save files or images) and real-time subscriptions. If sticking strictly to Replit's stack, Replit will provision a **Neon Postgres** database automatically [13] , which we can use with SQLAlchemy or Prisma. In either case, a well-defined schema will include tables like Users, Families (groups), Memberships, Messages, etc. to organize our data. We will also enable the **pgvector** extension (or use a separate vector DB) to store embeddings for semantic search in conversation history – this powers the long-term memory retrieval by similarity search on vectors. Regular backups and security rules (e.g. row-level security for multi-tenant data separation) will be applied to protect user data.
Memory Store (Vector DB)	**Vector database or embedding index** for long-term memory. To implement memory retrieval, we will encode important text (e.g. facts from conversations, user profile info, past queries) into vector embeddings and store them. Solutions include using **ChromaDB** or **Weaviate** (if self-hosting a vector DB) or simply storing vectors in Postgres pgvector if using Supabase/Neon. This allows the agent to perform similarity searches to find relevant past information. For quick lookup of recent context or conversation summaries, the system might also use an in-memory store (Redis or a Python cache) alongside the vector DB. A **memory management module** will periodically summarize older chat logs to keep the conversation history concise, storing summaries in the database so the agent can recall long dialogues without re-reading every message.

Layer / Component	Technology & Description
User Authentication & Profiles	**Supabase Auth** (if using Supabase) or a custom JWT-based auth system to handle user sign-up/login. Email invitation to family chat can be implemented via Supabase's invitation features or by generating a one-time token link. Upon registration, each user gets a profile entry in the database (with fields like display name, avatar URL, preferences JSON, etc.). We will secure all API endpoints so that only authenticated users can access their data or their family chat. Passwords (if used) are hashed; alternatively Magic Link or OAuth can be offered for convenience. Managing family groups means implementing a relationship between users – one user can create a "family room" and invite others, which the system represents via a group ID or separate table mapping users to group. The auth system should support at least basic roles or tags (e.g. to possibly distinguish an **adult** account vs a **child** account, if we want to tune content filtering or permissions in the future).
Agent Orchestration & Tools	**LangChain** and **LangGraph** frameworks to enhance agent capabilities. LangChain provides a library of **tools** (Google search, calculators, etc.) and an easy way to manage the agent's prompt decision-making. We can integrate LangChain to give Tongkeeper abilities like calling a calculator for math homework or fetching data from a knowledge base. **LangGraph**, built on LangChain, can enable more advanced multi-agent workflows – e.g. one sub-agent handling scheduling while another handles Q&A, coordinating their outputs [14] [15]. This could be useful as Tongkeeper grows (for instance, a dedicated "calendar agent" and a "tutoring agent" working together). By structuring the backend to be modular, we ensure new agents or tools can be plugged in without disrupting the whole system. Suna's design already uses an agent-manager approach (with the backend orchestrating tool use via the Docker sandbox [10] [6]); we will build on that to allow easy extension.
Deployment (Replit & Cloud)	**Replit Agents platform** will be used to deploy the initial version. Replit will provide a container (Linux environment) where we can run the FastAPI server (and possibly the Next.js frontend if we use a single instance approach). Replit also auto-provisions a Neon Postgres DB [13] and supports hosting web services with public URLs. We'll containerize the application for consistency – e.g. using Docker Compose similar to Suna's setup (one service for backend, one for front, etc.). On Replit, we might combine them or use a single Flask/WSGI to serve both API and static files for simplicity. The initial cloud deployment ensures we can quickly iterate and all family members can access Tongkeeper via the web. Over time, if scaling is needed, we can move to a dedicated cloud provider (AWS, GCP, etc.), using Kubernetes or separate services for each component (e.g. a managed Postgres, Redis, etc.). The architecture is cloud-agnostic enough that migrating off Replit would be straightforward when needed.

Table 1: Summary of key technical components for Tongkeeper's implementation, mapping features to specific technologies.

Each component above aligns with Tongkeeper's goals and leverages **open-source or proven frameworks**. Notably, adopting Suna's architecture (FastAPI + Next.js + Supabase + Docker) provides a solid baseline [10] [12], and we have tailored it to Tongkeeper's family-oriented use cases (adding multi-user chat and scheduling). The stack emphasizes flexibility – for example, using an LLM integration layer means we can start with DeepSeek and later **swap in a different model** without changing the front-end or conversation logic, much like Nextcloud's assistant which allows multiple LLM backends and even local hosting for privacy [16].

Future Enhancements and Extensibility

While the initial implementation focuses on text-based chat and core features, Tongkeeper's design anticipates future improvements:

- **Voice Input/Output:** A high-priority future update is adding **voice interaction**. This involves integrating Speech-to-Text (STT) and Text-to-Speech (TTS) modules. For STT, we could use an API like Google Cloud Speech or open-source alternatives (e.g. Whisper) to convert a user's spoken words into text for the LLM. For TTS (having Tongkeeper speak responses), services like Amazon Polly or Coqui TTS could generate natural voice audio. The frontend would gain a microphone button to record queries and a speaker icon to play the assistant's spoken response. The system design is ready for this: the back-end can handle audio inputs (perhaps as an upload or via streaming) and the front-end can handle audio playback. We will ensure the UI remains simple when voice is added – e.g., using familiar push-to-talk UX (like Siri) so iOS users feel at home. Voice capability will make Tongkeeper even more accessible (for kids who can't type or when users are multitasking).
- **Alternate and Multiple LLM Support:** Tongkeeper will remain **LLM-agnostic** to benefit from rapid advances in AI. The integration layer (via LiteLLM or LangChain) will allow connecting to new models or even using multiple models in tandem. For example, a future version might use a smaller on-device model for quick responses or sensitive data, and a larger cloud model for complex tasks. We might also integrate specialty models: if Google Gemini or other multimodal models become available, Tongkeeper could leverage them for tasks like image understanding (not in initial scope, but possible later). The key is that our architecture's decoupling of "agent logic" from the specific LLM makes this feasible. Configuration could allow an advanced user to plug in an OpenAI API or local Llama2 model as the backend, for instance. This future-proofing ensures Tongkeeper stays state-of-the-art without a complete rewrite.
- **Hybrid Deployment Options:** In the long term, we plan to support a **hybrid cloud-edge architecture**. For privacy-conscious families, Tongkeeper could be deployed on a home server or personal device (running the core agent locally, so data never leaves the home), while still optionally connecting to cloud services for heavy tasks or data sync across devices. Our use of open-source components (Postgres, Docker, etc.) means a tech-savvy user could self-host Tongkeeper fully. We will continue to optimize for lower resource usage so that a future lightweight version of the agent (perhaps using a smaller LLM like Mistral 7B or a quantized model) can even run on a laptop or mobile device. This also adds resilience – the assistant remains available even if internet is down (for local tasks like answering a homework question using cached knowledge). The cloud portion can be used for updates, sharing data between family members, or performing large computations on demand.
- **Enhanced Tool Integrations:** Tongkeeper's capabilities can grow by adding more **plugins/agents**. With the architecture in place, we can integrate additional tools: e.g., a **calendar sync agent** that links with Google Calendar or Apple Calendar so that events created in Tongkeeper reflect on users'

actual calendars; a **smart home agent** that connects to IoT devices (letting users say "Tongkeeper, turn off the kitchen lights"); or an **email agent** that can draft and send emails on command. We envision using frameworks like LangChain's tool abstraction to plug these in seamlessly. The multi-agent coordination possible with LangGraph means Tongkeeper could internally delegate tasks – for instance, if a query is about scheduling, it could hand off to a scheduling sub-agent, whereas a general knowledge query goes to a QA agent. This collaboration among agents would happen behind the scenes, keeping the user experience unified [14]. The result is **extensible functionality**: developers could contribute new skills to Tongkeeper (given it's based on open source Suna, community contributions are possible) in a plugin-like fashion.

- **Mobile and Cross-Platform Clients:** Although the initial UI is web-based, we consider future development of **mobile apps** (iOS/Android) or at least a progressive web app. The iOS design familiarity will make it easier to eventually wrap the web app into a native-like app (using frameworks like React Native or Capacitor). A native app could also enable push notifications for reminders and voice input with less friction. Similarly, a desktop application or integration (like a menu-bar app on macOS for quick queries) could be explored. The backend's API-centric approach means any client can interface with Tongkeeper easily, so adding new frontends won't require major changes to the core. Ensuring the web app is responsive and touch-friendly from the start will pave the way for these options.
- **Security & Privacy Enhancements:** As a family-oriented assistant, future versions will double-down on **privacy controls**. We plan to allow local-only modes (as mentioned) and provide transparency (users can review what data is stored in their profile or chat history). Content filtering will also be tuned especially if kids use the system – possibly incorporating an open-source moderation model to filter or rephrase inappropriate content. End-to-end encryption of chat data is another possible enhancement, especially if data is stored in the cloud – ensuring that even the service operators can't read conversation logs, which might appeal to users who are concerned about sensitive family discussions. These are not in the MVP, but the architecture's reliance on open-source and self-hostable components aligns with giving users **full control** over their data and trust in the system (similar to how Nextcloud's assistant focuses on privacy by allowing self-hosting [17] [16]).

By implementing the above plan, Tongkeeper will start as a powerful personal/family AI assistant with robust scheduling, task help, and collaborative chat features, and it will have the **technical foundation to evolve** – integrating new AI advancements (voice, better models) and expanding its utility in a safe, user-friendly manner. With a well-chosen tech stack and modular design, Tongkeeper can grow from a cloud-based chatbot into a comprehensive family AI companion.

Sources:

1. Kortix Suna – *Open Source Generalist AI Agent* (Architecture & Tech) [10] [12]
2. Guo, Lewis. *Suna AI: The Open-Source Generalist Agent…* – Medium, May 2025 (architecture insights, real-time and data layers) [18] [9]
3. DeepSeek LLM – Open-Source 67B Language Model (GitHub README) [1]
4. *OpenAI's ChatGPT adds scheduled tasks feature* – The Verge (industry context for reminders) [2]
5. LangChain/LangGraph Documentation – multi-agent orchestration capabilities [14] [15]
6. Neon Tech Blog – *Replit Agent and databases* (Replit's Postgres/Neon provisioning) [13]
7. Nextcloud AI Assistant Announcement – (on multi-LLM flexibility and privacy) [16]

1 GitHub - deepseek-ai/DeepSeek-LLM: DeepSeek LLM: Let there be answers
https://github.com/deepseek-ai/DeepSeek-LLM

2 OpenAI's ChatGPT adds scheduled tasks feature in beta | The Verge
https://www.theverge.com/2025/1/14/24343528/openai-chatgpt-repeating-tasks-agent-ai

3 4 GitHub - kortix-ai/suna: Suna - Open Source Generalist AI Agent
https://github.com/kortix-ai/suna

5 9 18 Suna AI: The Open-Source Generalist Agent Revolutionizing Task Automation | by Lewis Guo | May, 2025 | Medium
https://medium.com/@guolisen_38580/suna-ai-the-open-source-generalist-agent-revolutionizing-task-automation-742f3f9f5fe7

6 7 8 10 11 12 Suna AI: the Open Source General AI Agent
https://apidog.com/blog/suna-ai-open-source-general-ai-agent/

13 Looking at How Replit Agent Handles Databases - Neon
https://neon.tech/blog/looking-at-how-replit-agent-handles-databases

14 15 Building AI agent systems with LangGraph | by Vishnu Sivan | The Pythoneers | Medium
https://medium.com/pythoneers/building-ai-agent-systems-with-langgraph-9d85537a6326

16 17 Meet the Nextcloud AI Assistant - Nextcloud
https://nextcloud.com/blog/first-open-source-ai-assistant/