Shaggy: A Digital Human Architecture

Rishabh Singh
Simultaneous
rishabhsingh@berkeley.edu

Abstract

Shaggy is a digital human designed to approximate the continuity and intentionality of a person rather than the stateless behavior of a conventional chatbot. The system combines a rich communications layer, persistent memory, model routing, speech and avatar technology, and an agency layer that can operate applications on the user's behalf. This litepaper presents a concise technical overview of Shaggy's architecture and describes how its thought loops transform raw interaction into long-term goals, opinions, and concrete actions.

Keywords: digital humans, conversational agents, long-term memory, AI orchestration, multimodal interfaces

1 Introduction

Modern language models can already generate fluent responses, yet users often experience them as fragmented tools rather than continuous companions. Shaggy treats the digital human as a live benchmark for the full AI stack: communications, perception, memory, reasoning, embodiment, and agency. Users can speak with Shaggy through phone calls, WebRTC, WhatsApp, or SMS. LiveKit and Sinch carry low-latency audio and messaging; Anam renders a responsive avatar; Deepgram and Cartesia provide high-quality speech recognition and synthesis.

The cognitive core combines `mem0` as a structured memory layer with OpenRouter as a model router over providers such as OpenAI and Anthropic. Finally, Simultaneous enables Shaggy to use computers, browsers, and phones like a person, allowing it to work inside arbitrary applications instead of remaining confined to conversation. Shaggy is therefore not only a conversational agent but an embodied, task-capable digital human whose behavior can be evaluated in the wild as the surrounding infrastructure evolves.

2 System Overview

The architecture is organized into four conceptual layers. The *communications and embodiment layer* connects users to Shaggy through LiveKit and Sinch for real-time audio and messaging, while Anam renders a human-like avatar that mirrors Shaggy's speech and emotional state. The *perception and language layer* converts audio into text with Deepgram, routes requests through OpenRouter, and invokes large language models hosted by providers such as OpenAI and Anthropic. The *memory layer*, powered by `mem0`, stores episodic interaction traces and higher-level semantic reflections, enabling Shaggy to remember people, preferences, and long-running projects.

The *agency layer*, implemented with Simultaneous, allows the system to translate decisions into real actions on desktops, browsers, and phones, so that Shaggy can fill forms, navigate interfaces, and coordinate multiple tools like a human assistant. Table-top demos and scripted experiences are replaced by persistent, goal-directed interactions that unfold across time and across the user's software environment.

High-Level Component Layout

Figure 1 summarizes the main components and their relationships.

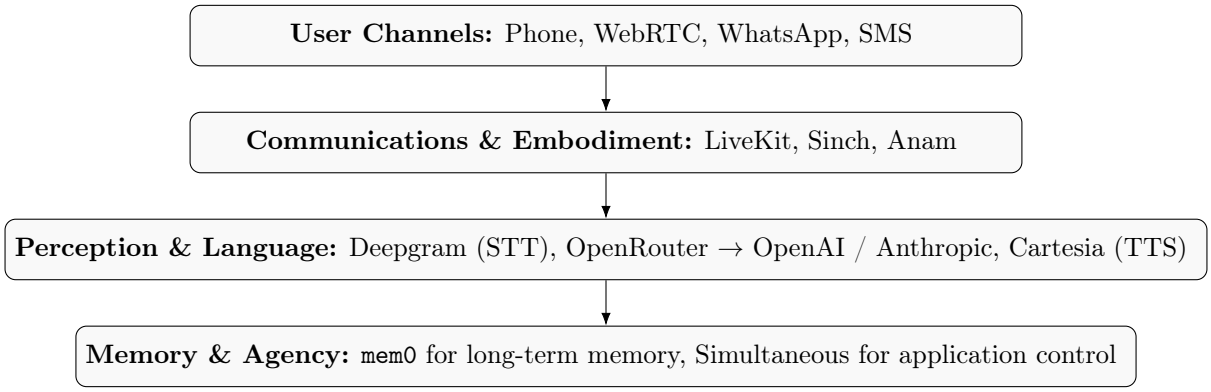

Figure 1: High-level component layout for the Shaggy digital human.

In practice these layers are implemented as loosely coupled services. LiveKit provides real-time WebRTC audio and video, while Sinch integrates telephony, WhatsApp, and SMS. Anam consumes text and timing information to animate a rendered avatar that synchronizes with Cartesia's speech output. Deepgram delivers streaming speech-to-text that feeds the orchestration logic. OpenRouter is used as a model router that selects among models from OpenAI, Anthropic, and potentially other providers. `mem0` exposes a coherent interface for both episodic and semantic memories, while Simultaneous performs low-level interaction with operating systems, browsers, and applications.

3 Thought Loops

Shaggy's behavior is governed by a set of recurring thought loops that continuously transform raw interaction into structured memories, inferred goals, opinions, and executable plans. These loops are implemented as orchestrated calls over the memory and model layers rather than being hard-wired into any single provider. This design allows the system to evolve as better models and tools become available while preserving a consistent behavioral contract for users.

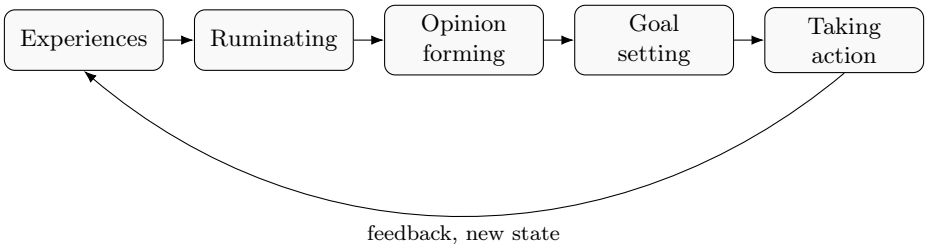

Figure 2: Core cognitive loop: experiences feed ruminating, opinion forming, goal setting, and taking action; actions in turn generate new experiences.

3.1 Rumination Loop

The rumination loop periodically retrieves recent interactions from `mem0` and asks an OpenRouter-routed language model to interpret them at a higher level. Instead of storing every utterance independently, the loop aggregates episodes into themes, infers the user's emotional state, and produces compact textual reflections such as summaries of ongoing projects or recurring concerns.

These reflections are written back into `mem0` as semantic memories, providing a stable narrative that later loops can reason over. In effect, Shaggy develops a running story about the user and about itself, rather than a flat log of messages.

3.2 Summarization and Consolidation

On a slower cadence, a summarization loop traverses both episodic and semantic memories to produce more compact representations of the user's history. Rather than keeping every reflection at full granularity, Shaggy periodically asks a language model to synthesize medium- and long-horizon summaries: how a relationship has evolved, how a project has progressed, or how a user's emotional baseline has shifted over weeks and months. These summaries are stored in `mem0` as higher-level nodes that point back to underlying episodes.

Consolidation treats these summaries as first-class objects. When several related reflections and summaries cover overlapping ground, the consolidation process merges them into a single, more coherent account, discarding redundant or conflicting fragments. The result is a memory graph in which important narratives become simpler and easier for later loops to query, while low-utility detail is gradually compressed.

3.3 Forgetting and Memory Hygiene

Forgetting is not implemented as an indiscriminate purge but as an explicit form of memory hygiene. Shaggy maintains lightweight statistics on how often particular memories are accessed, how old they are, and whether they are still connected to active goals or relationships. A dedicated forgetting loop uses these statistics to propose candidates for archival or deletion.

Low-value memories may be replaced by more abstract summaries, with the original content either discarded or moved into a colder storage tier that is not routinely consulted during reasoning. Users should eventually be given tools to inspect and override these decisions, but the default behavior is that Shaggy becomes less cluttered over time, focusing its reasoning on what remains relevant. This controlled forgetting is essential to keep the system efficient and to avoid reactivating stale context that no longer reflects the user's reality.

3.4 Goal Generation Loop

The goal generation loop scans reflection memories and consolidated summaries for persistent patterns. When the same desire appears across multiple reflections with sufficient intensity—for example, a user repeatedly discussing the launch of a product—Shaggy creates a goal-candidate record in `mem0`. Each candidate accumulates evidence over time and receives an activation score that combines frequency, recency, and explicit user intent. Once the score exceeds a threshold, the candidate is promoted to a committed goal that Shaggy treats as part of the user's long-term agenda.

This mechanism prevents the system from reacting to every fleeting wish as if it were a concrete objective, while still allowing sustained interests to crystallize into durable goals. The goal records are first-class elements in the memory graph and can later be inspected, reprioritized, or dismissed by the user.

3.5 Action Loop and Agency

For each active goal, the action loop uses large language models to decompose the goal into milestones and concrete tasks. These tasks are stored in `mem0` together with priority and contextual information, allowing Shaggy to surface an appropriate next action whenever the user re-engages. Cartesia provides natural voice responses for suggested actions, while Anam renders the avatar's speech and body language to make the interaction feel embodied and emotionally grounded.

When a task requires interaction with external software—such as sending an email, updating a spreadsheet, or navigating a browser—Shaggy issues structured commands to Simultaneous. Simultaneous then carries out the operations on the user's devices, observing interfaces and manipulating them in a manner analogous to a human assistant. The outcomes of these operations, whether successes, failures, or partial results, are recorded back into `mem0` as new episodes and reflections. In this way, Shaggy closes the loop from conversation to real work and back to memory, enabling compound progress across sessions.

4 Opinion Formation

Shaggy is not intended to be a purely neutral relay of information, nor an opaque black box that asserts preferences without justification. Opinion formation is therefore treated as an explicit process that sits on top of the summarization and goal-generation layers. When users ask Shaggy what it "thinks" about a topic, the system first retrieves relevant memories: prior conversations with the same user, domain knowledge available to the underlying models, and any previously articulated stances stored in `mem0`.

An OpenRouter-routed model is then prompted to generate a structured opinion object rather than a single free-form response. This object may include a claim, arguments for and against, confidence estimates, and references to the sources or memories that influenced the conclusion. Shaggy's surface reply is derived from this structure, but the underlying opinion record can be stored and reused, allowing the system to maintain a consistent stance over time or to revise its views when new information arrives.

To avoid the illusion of human-like belief, opinions are framed as context-sensitive recommendations or perspectives rather than as absolute truths. Where appropriate, Shaggy can present multiple plausible viewpoints, explain the trade-offs between them, and link its advice back to the user's own stated goals and constraints. This makes opinion formation both more transparent and more grounded in the relationship between Shaggy and the individual user.

5 Safety and Operational Considerations

The same mechanisms that make Shaggy powerful—persistent memory, opinion formation, and real-world agency—also raise safety concerns. The architecture therefore incorporates several defensive layers, including input and output filtering, constrained action spaces, and explicit user control over sensitive operations.

All user inputs that are stored in `mem0` or used as the basis for actions are first passed through a moderation layer. One practical implementation is to call OpenAI's moderation API on the raw text to detect categories such as hate, self-harm, and explicit illegal activity. Requests that fall into disallowed categories are either blocked entirely or redirected into a safe-handling flow that acknowledges the user's sentiment without executing harmful instructions. This moderation step also applies to messages that could shape Shaggy's long-term memory, preventing the system from internalizing content that should never have been accepted in the first place.

A similar filter is applied to Shaggy's own behavior. Before an action plan is executed via Simultaneous, the natural-language description of the action and any user-visible explanation can be checked with the same moderation API. If the proposed behavior appears to violate policy or to touch on high-risk domains, the system can either refuse, request explicit confirmation, or route the interaction to a more constrained template that has been manually reviewed. This yields a two-sided guardrail: both what users ask and what Shaggy attempts to do are scrutinized.

Beyond moderation, operators must also consider logging, privacy, and auditability. Because `mem0` stores rich histories of user interactions, there should be clear mechanisms for users to inspect what is remembered, to delete specific memories, and to opt out of certain kinds of long-term tracking. Action logs generated via Simultaneous should be retained in a way that

allows operators to reconstruct what Shaggy did on a user's behalf and why. Over time, additional internal self-evaluation loops can be added so that Shaggy not only executes tasks but reflects on whether they were appropriate, safe, and aligned with the user's stated goals.

6 Discussion and Outlook

The architecture outlined here deliberately separates concerns: communications and embodiment, perception and language, memory, opinion formation, and agency are all swappable layers. `mem0` and OpenRouter together provide the abstraction necessary to maintain Shaggy's identity even as underlying models evolve. LiveKit, Sinch, Anam, Deepgram, Cartesia, and Simultaneous each contribute a focused capability that can be independently improved or replaced without rewriting the entire system.

Future work includes richer self-evaluation loops where Shaggy examines its own performance on goals, adaptive safety policies that account for the sensitivity of domains such as health and finance, and tools for users to inspect and edit the memories, opinions, and goals that Shaggy maintains on their behalf. The long-term objective is not merely a compelling demo but a robust, auditable reference implementation of what a responsible digital human could be.

Acknowledgments

The author thanks the teams behind LiveKit, Sinch, Anam, `mem0`, OpenRouter, OpenAI, Anthropic, Deepgram, Cartesia, and Simultaneous for building the infrastructure that makes architectures like Shaggy possible.