Case Study 1 Report

Gleb Vlasov
CS553-S26

Note: Headings take a lot of space. I promise it's not that long. I already shortened it by over a page. I just made a section for every item mentioned in the assignment.

1 Product Proposal

1.1 Description

The Product developed in our Case Study is an LLM chatbot that captures "Memory Notes" from users' messages. A Memory Note is a simple, structured JSON object that stores a concise fact about the user extracted from the conversation.

Additionally, the LLM can operate under multiple "Personalities" selected by the user or deployer. A Personality constrains what information is gathered and guides the conversation toward collecting more relevant facts within its scope, rather than recording identifiable information.

1.2 Purpose

The purpose of our Product is to provide a condensed, queryable context history for a user. The resulting memory set can support user-focused research and improve chatbot usability.

By providing a lightweight interpretation of the user, the product enables a more dynamic approach to data collection. It can act as a more approachable alternative to questionnaires, encouraging more honest answers through frictionless interaction. Using Personalities, a research team can focus the AI on relevant information and increase the yield of useful entries by guiding the user.

A second major application is improving recall in long-running conversations. As chat history grows, recall deteriorates, and inputs may be truncated to fit within a context window. In sensitive applications such as everyday assistants or mental health analysts/counselors, losing important information or trends can be critical. Instead of sending full history (slow, costly, and hard-capped), Memory Notes are compact and can emphasize the most relevant details.

1.3 Target Audience

The target audience is chatbot developers supporting long-running chats and sensitive topics that require stronger recall, and researchers exploring frictionless methods of data collection.

2 Technology Stack (models)

2.1 API Model

The API deployment uses openai/gpt-oss-20b via the Hugging Face Inference API (chat completion). OpenAI describes it as a Mixture-of-Experts Transformer model intended for low latency, with support for configurable reasoning effort and agentic/tool-use capabilities.

The model is text-only and supports a large context window. Hugging Face documentation notes it is meant to be used with OpenAI's Harmony response format. In this product, it serves as the higher-capability remote option when the user selects API mode and is sufficient for a rudimentary showcase without exhausting free-tier limits.

Public documentation does not provide a concrete list of training datasets, but the model card describes the training approach as large-scale distillation and reinforcement learning, and emphasizes the required rendered chat/Harmony format.

2.2 Local Model

The local deployment uses microsoft/Phi-3-mini-4k-instruct via a Transformers text-generation pipeline. Phi-3 Mini-4K-Instruct is a dense decoder-only Transformer with 3.8B parameters and a 4K token context length, trained on an offline dataset with a cutoff date of October 2023. Microsoft states it is fine-tuned with SFT and DPO for instruction-following and safety alignment.

In this product, Phi-3 provides the offline inference mode when the user selects local mode.

3 Performance Analysis

3.1 Response Time

API calls are consistently fast, typically below ~0.8 seconds on average and sometimes around 0.3 seconds. Overall API delay feels minimal. The local version performs far worse. Early in development, it took roughly a minute per response; as prompt complexity increased, latency grew dramatically. The current version can take around 7 minutes per response on a high-end laptop CPU, and can be slower on a Hugging Face Space. Reducing prompt complexity can improve local speed, but risks reducing output quality and rule-following.

3.2 Error cases

Two major error cases are empty responses and parsing failures. While both were initially frequent (roughly every other request), iterative prompt work plus message-cleaning/formatting and edge-case handling reduced the error rate to roughly 1–2 failures in 30+ turns. Further improvements are likely by providing more precise memory usage information in requests.

3.3 Memory Quality

The AI handles entry creation well and identifies relevant information frequently (roughly 70% of the time). Quality deteriorates when the conversation spirals into a single niche, narrowing the AI’s focus.

Personality performance varies. The AI tends to follow its scope when the conversation begins inside it, but focus degrades if the conversation starts outside its intended scope. Memory included in the request significantly affects performance; sampling or prioritization strategies should improve results.

The memory note format and label consistency were strong: outputs were concise and informative, with good consistency. Importance ratings were inconsistent across runs and were affected by prompt changes; including importance in the request payload also degraded quality.

4 Cost Analysis

4.1 Plan and Rate Limits

The application requires login via Hugging Face, so rate limits and billing follow the user’s account rather than the developer’s. With Hugging Face Inference Providers, free users receive limited monthly credits, which can cover a few hundred to around a thousand turns depending on the provider and settings.

When “Use local model” is enabled, throughput is constrained by Space hardware. On the free plan used for this Case Study, the Space has 16GB RAM and 50GB non-persistent disk. Free Serverless Inference API usage is rate-limited (on the order of a few hundred requests per hour), and rate limiting can temporarily prevent API-mode usage.

4.2 Estimated Cost

4.2.1 API Cost

With user login billing, hosting costs are primarily Space runtime while inference costs are distributed across users based on their Hugging Face plan and usage. With centralized billing, inference cost scales linearly with usage (requests * tokens * provider pricing).

Assuming 100 turns per user and 1,000 users, default settings require ~51M input/output tokens. Pricing varies by provider and scales directly with token volume.

4.2.2 Local Model (Space hosting)

Local inference cost is driven by Space hardware rather than tokens. Hardware tiers range from CPU to GPU, and higher concurrency requires larger hardware and/or more Spaces.

4.3 Optimization

Current cost controls include:

- Bound output size: lowering max_tokens reduces generated tokens and computation for both API and local inference.
- Limit input context: sending only the last N turns reduces input size and token usage.
- Reduce memory injection: the memory importance filter limits how many memory items are appended to prompts.

Further constraints reduce quality, so the most practical optimization is selecting an inference provider carefully and choosing a primary focus on either local or API inference.

5 Concerns

5.1 Privacy

- Profile storage: user memory is stored server-side in a dictionary keyed by user_id and personality. It is not persisted, but exists in RAM for the lifetime of the instance.
- Memory as context: memory items are included in requests as context and are sent to the backend API.

5.2 Security

- Prompt injection: the user can prompt the model to write malicious or incorrect "memory." Structure is validated, but truthfulness is not verified. Tools to edit memory could partially mitigate this.
- Authentication: OAuth avoids embedding a shared token, but ties API usage to the user's limits/billing and depends on the user logging in.
- Content moderation: the app does not include prompt moderation or filters, so misuse is possible.

5.3 Scalability

- Local mode is hardware-bound: the local pipeline is loaded once and kept in RAM for the process lifetime; Space constraints limit throughput/latency and Gradio queuing. Higher traffic requires larger hardware and/or a different deployment.
- Memory growth is unbounded: memory grows over time and can become a RAM issue for many users or long sessions.
- API mode has limits: availability, rate-limits, and credit limits can throttle usage