Cyberlife AI: A No-Code Platform for Building Memory-Enhanced Conversational Agents with RAG

MCDS Capstone Project Final Report

Webber Wu[1] **Eagle Lo**[1] **Brad Chen**[1]

[1]Carnegie Mellon University, Language Technologies Institute

{guanweiw, eaglel, potingc}@andrew.cmu.edu

CyberLife Repositories **qinchuanhui/UDA-QA (Ref)**

Abstract

This report presents CyberLife AI, a no-code platform for creating personalized, topic-driven conversational agents with persistent memory and knowledge grounding. We address limitations in existing platforms through a memory-augmented dual-retrieval RAG pipeline combining conversation history recall with document-based knowledge retrieval. The system enables non-technical users to create agents by uploading domain-specific documents, defining personality traits, and customizing voice and appearance through integrated speech recognition, text-to-speech synthesis, and animated avatar generation. The architecture comprises a TypeScript/React frontend, Python Flask backend, and separate GPU model server for avatar generation.

1 Introduction

Modern conversational AI platforms like Replika and Character AI restrict user customization, lack persistent memory, and provide minimal support for domain-specific knowledge. These limitations prevent educators, creators, and small businesses from building agents that reflect their expertise and communication style without programming experience.

Cyberlife AI is a no-code platform for building topic-driven conversational agents that combine natural dialogue, consistent personality, and long-term contextual understanding. Users define an agent's identity and upload personal logs, blogs, or research papers, which the platform automatically indexes for retrieval-based reasoning. Unlike traditional chatbots that rely on short-term context, our system enables highly personalized agents that can index, recall, and reason over numerous private documents.

Our core innovation is a **dual-retrieval RAG pipeline** that strengthens contextual reasoning through two specialized retrievers: one retrieves semantically relevant dialogue history from the agent's memory, maintaining conversational continuity, while the other accesses external creator-supplied documents for factual grounding. When local knowledge is insufficient, the system falls back to web search. This hybrid mechanism, inspired by research in retrieval-augmented generation (RAG) (Lewis et al., 2020), neural facial animation (Kerbl et al., 2023), and emotion-aware voice synthesis (Casanova et al., 2022), ensures responses are both personally coherent and domain-accurate. For example, a Spanish tutor can create an AI assistant that mirrors her teaching style, remembers previous lessons, and provides tailored feedback without code.

2 Hypothesis/Project Goal

We hypothesize that a memory-augmented dual-retrieval framework enables non-technical users to create topic-driven conversational agents that maintain both conversational coherence and factual grounding. By combining dialogue history recall with document-based knowledge retrieval, agents can sustain natural, contextually rich interactions within specific domains.

Our goal is to demonstrate that this dual-retrieval design, where one component recalls relevant past exchanges and another accesses user-supplied documents, produces agents that are coherent, knowledge-driven, and adaptive over time. This positions Cyberlife AI as a foundation for "personal log" chatbots capable of meaningful, long-term dialogue grounded in users' real data. Through evaluation on the topic-specific dataset using ROUGE, BLEU, and BERTScore metrics, we aim to validate that conversational agent creation can become more inclusive, scalable, and contextually aware.

3 Relationship to Prior Work

Recent advancements in large language models (LLMs) have enabled the creation of increasingly capable conversational agents. Systems such as ChatGPT and Claude rely heavily on massive pretrained transformers with instruction tuning, allowing general-purpose dialogue without task-specific training. However, these systems often lack persistent memory across conversations and are typically grounded in static, global knowledge with limited personalization.

To address these shortcomings, RAG has emerged as a promising paradigm. Pioneering work like REALM (Guu et al., 2020) and RAG (Lewis et al., 2020) introduced architectures that combine LLMs with a document retriever to dynamically fetch context-relevant information. While these methods significantly improve factual grounding, they primarily focus on retrieving from global corpora (e.g., Wikipedia) and do not support long-term dialogue memory or personalized knowledge bases.

More recent system like MemGPT (Packer et al., 2023) has explored persistent memory in conversational agents. MemGPT uses structured memory buffers and task-based memory management to help agents recall prior user interactions. However, these approaches typically require extensive system-level control or fine-tuning, limiting their accessibility to non-technical users.

In parallel, low-code and no-code platforms such as Character.AI and Inworld have enabled users to create agent personas through prompt engineering and personality scripting. These platforms, while user-friendly, do not offer mechanisms for agents to learn from user-uploaded documents or interact meaningfully with personalized knowledge over time.

Our work, **Cyberlife AI**, aims to bridge these gaps by integrating a dual-retrieval system that supports both long-term dialogue memory and user-specific document grounding. Unlike prior RAG systems that retrieve from fixed external corpora, our approach enables each user to upload their own documents such as personal logs, reports, notes, and domain-specific content, which are dynamically indexed into a scalable vector store for real-time retrieval. In doing so, our platform allows users to build topic-specialized agents that can not only recall their own conversational history, but also reason over their own data.

Moreover, we emphasize a no-code interface that lowers the barrier for customization, making memory-augmented conversational AI accessible to non-developers. This combination of personalized RAG, memory, and usability differentiates our system from existing research and commercial platforms.

Through this work, we aim to show that combining dual retrieval, modular memory, and user-centric design produces agents that are more helpful, coherent, and contextually aware, especially in personal or domain-specific applications.

4 Spring/Fall Semester Development Goals

At the beginning of the semester, our primary goal was to build a working prototype of a no-code platform for creating topic-specific conversational agents. We aimed to enable users to define an agent's persona and upload domain-specific documents to support more grounded conversations. The initial milestones included building a frontend interface, integrating a backend LLM API, and implementing basic agent creation functionality.

By the midpoint of the semester, after receiving feedback from both our mentor and course instructor, we shifted our focus toward designing a more technically robust and research-oriented system. This pivot emphasized three updated goals:

1. **Develop a dual-retrieval RAG pipeline**: One retriever focuses on dialogue history and the other on user-uploaded documents, with fallback to a web search API.

2. **Implement scalable document ingestion and memory architecture**: Enable agents to process large volumes of user-provided data, such as notes, blogs, and PDFs.

3. **Prepare an evaluation framework using real-world datasets**: Design experiments using topic-specific datasets to evaluate grounding quality and memory relevance.

Milestones Achieved:

- We implemented the full dual-retrieval pipeline, including vector databases for both conversation history and user knowledge, and added a web search fallback mechanism.

- We integrated PDF ingestion and successfully processed a variety of real documents, validating that the system can support personal knowledge bases.

- SERP bugs and prompt formatting issues were resolved, leading to more reliable responses.

- A working end-to-end demo was prepared, integrating frontend and backend workflows.

Ongoing Work:

- We are finalizing the experiment infrastructure to evaluate the system using the topic-specific dataset.
- Frontend polish and robustness improvements are planned as part of the final sprint.
- Poster and documentation preparation is underway for final presentation and handoff.

This shift from a simple integration project to a deeper system and research focus has helped us build a more scalable and meaningful product. The goals achieved this semester lay the groundwork for richer memory-augmented agents and experimentation in the next phase.

5 Project Requirements

5.1 Intended Users and Usage Overview

The system is built for two main groups: creators, who provide the materials that the agent can draw from, and end users, who interact with the agent through natural language.

Creators. Creators supply the documents and content that form the system's retrieval base. They upload text sources, define character persona attributes, or provide reference materials that will later be indexed and used during conversation. Their role focuses on preparing data rather than managing complex configuration.

End users. End users communicate directly with the agent. They expect the system to recall relevant information from past interactions or from the materials provided by the creator, and to use that information to answer questions or continue the conversation consistently.

Feature differentiation. Creators contribute the underlying resources such as knowledge base documents and avatar materials, while end users rely on the system to produce accurate retrieval and coherent responses. The workflow is designed to keep data preparation simple for creators and to provide a seamless conversational experience for end users.

5.2 System Functionality

The platform currently supports the following functional capabilities:

- Agent creation through a no-code web interface.
- Persona customization (name, character traits, image, voice).

- Document ingestion from plain text and PDFs.
- Semantic search over user-uploaded documents and conversation history.
- Dynamic fallback to web search when local data is insufficient.
- Real-time, chat-based interaction with memory-augmented agents.

5.3 Non-Functional Requirements

Key non functional requirements include:

- **Scalability**: The system should support large per user datasets (e.g., 100+ documents) and operate across thousands of user agents.
- **Responsiveness**: Text based interactions should return within 3 seconds, and animated character responses should finish within 10 seconds to maintain a smooth user experience.
- **Security and privacy**: User data, including private documents, must be securely stored and accessible only to the creator's agents.
- **Modularity**: Backend components such as dialogue retrieval, document search, and web search should remain decoupled to enable flexible experimentation and updates.

5.4 Resource Requirements

To support current and planned system functionality, the platform requires:

- **Compute**: Cloud servers with GPU acceleration for video generation and embedding computation.
- **Storage**: Persistent storage for uploaded documents, vector embeddings, and conversation histories.
- **LLM API access**: High capability language models (Gemini 2.5 Flash (Comanici et al., 2025) and Gemini 2.0 Flash (Google DeepMind Team et al., 2023)) with context windows large enough to support retrieval outputs and system prompts.

- **Web search API**: An integrated third party search service (SerpAPI) to provide fallback retrieval when local knowledge is insufficient.

These requirements ensure that Cyberlife AI remains usable, secure, and extensible as we move toward supporting richer use cases and more sophisticated agent interactions.

6 Experiment/System Design Overview

6.1 System Architecture

[Will put the system architecture chart in the final report]
CyberLife AI follows a three-tier microservices architecture designed for scalability and separation of concerns. The system consists of:

- **Frontend Layer**: A TypeScript/React application deployed on Vercel, providing the user interface for agent creation and interaction. The frontend implements a router-based navigation system with dedicated pages for landing, avatar creation, blog content, and real-time conversation.

- **Application Backend**: A Python Flask server managing the main logic, conversation state, and API orchestration. This layer handles speech-to-text conversion, retrieval-augmented generation, memory management, and coordinates between various subsystems.

- **Model Server**: A dedicated GPU-enabled server running SadTalker models (Zhang et al., 2023) for avatar animation. This separation prevents GPU operations from blocking the main application and enables independent scaling.

6.2 Core Subsystems

6.2.1 Dual-Retrieval RAG Pipeline

Our system integrates two primary retrieval pathways, supported by a fallback web search mechanism. Together, these components allow the agent to draw from past conversations, user-provided documents, and external information sources.

Conversation memory retrieval. We provide two methods for accessing relevant content from prior interactions:

- **Simple history**. Maintains a sliding window of recent conversation turns for each session, ensuring immediate local context.
- **Embedding based retrieval**. Encodes all historical conversations as dense vectors using Sentence-BERT (Reimers and Gurevych, 2019) (all-MiniLM-L6-v2). At query time, the user's question is embedded and the top-K semantically similar exchanges are retrieved using FAISS (Douze et al., 2024) approximate nearest neighbor search, enabling long term memory beyond the local window.

Knowledge base retrieval. For user uploaded documents, we apply a lightweight document processing and embedding pipeline:

1. Extract text from PDFs using PyPDFtwo.
2. Split text into overlapping chunks and embed each chunk using the same encoder.
3. Store all embeddings in a FAISS index and retrieve the top-K relevant chunks at query time for inclusion in the prompt.

Web search fallback. If neither conversation memory nor the knowledge base provides sufficient information, the system issues a web query via SerpAPI to obtain up to date external content.

6.2.2 Agent Persona System

The CompanionAgent class implements a structured persona framework. User-provided metadata (name, usage type, description, character traits) is compiled into a prompt template:

`"Your name is {name}, and your usage type is {usage}, which means {description}. You should have the characteristics of {character}."`

This persona string is prepended to every LLM prompt, ensuring consistent personality across conversations.

6.2.3 Multimodal Response Generation

The response pipeline follows these steps:

1. **Speech-to-Text**: User audio is captured via WebRTC MediaRecorder, converted to WAV format using FFmpeg, and transcribed using Google Speech Recognition API.
2. **Context Assembly**: Relevant conversation history and knowledge chunks are retrieved and concatenated with the user's transcribed query.
3. **LLM Generation**: The assembled context is sent to Gemini 2.0 Flash (Google DeepMind Team et al., 2023) with the agent's persona prompt along with retrieved information.
4. **Text-to-Speech**: The response is synthesized using Coqui TTS (Casanova et al., 2022).
5. **Avatar Animation (Optional)**: If video mode is enabled:
   - The audio and source image are sent to the Model Server
   - SadTalker (Zhang et al., 2023) generates 3DMM coefficients from audio
   - Face render pipeline produces animated video
   - Video is returned to frontend for playback

6.3 Data Flow

[We are currently updating the system design this semester. The flow chart will be finalized and included in the final report.]

1. The user records an audio message.
2. The frontend sends the audio file to the backend.
3. The backend performs speech to text conversion.
4. The system retrieves relevant conversation history.
5. The knowledge base is queried for relevant document chunks.
6. The conversation history, persona information, and retrieved documents are sent to the Gemini API.
7. The generated response is appended to the conversation history.
8. Text to speech converts the response into audio.

9. (Optional) The model server generates a video.

10. The audio or video output is returned to the frontend for playback.

6.4 Key Design Decisions

6.4.1 Stateless Backend with Session Management

Rather than maintaining in-memory session state, we use Flask sessions with secure cookies. Each user receives a session_id that keys into conversation history and vector stores.

6.4.2 Model Server Isolation

By separating the GPU-intensive SadTalker inference into a dedicated service, we achieve:

- Non-blocking main application (I/O-heavy operations don't wait for GPU)
- Ability to run model server on GPU instance while app backend runs on CPU instance

6.4.3 Embedding Model Selection

We chose `all-MiniLM-L6-v2` for embedding generation because 384-dimensional vectors should be enough to balance expressiveness and storage.

7 Experimental Design

7.1 Dataset

We evaluate our conversational memory retrieval system using the UDA-QA dataset (Hui et al., 2024), a benchmark designed to assess question answering capabilities over diverse document types. The dataset includes PDF documents categorized into three primary fields: Finance, Academic Papers, and Wikipedia articles. Each document is paired with a set of questions and corresponding ground truth answers that require information retrieval from the source material. The dataset structure allows us to test retrieval performance across different document domains and complexity levels. The UDA-QA instances are organized with document metadata that specifies the field category along with question-answer pairs. Each question is designed to test the system's ability to locate and extract relevant information from the corresponding PDF document.

7.2 Machine Learning Models and Pipeline

Our experimental pipeline consists of four main stages. First, we process the PDF documents from the UDA-QA dataset to serve as the knowledge base for our retrieval system. Each PDF is treated as user-uploaded content that the system must index and later retrieve from. Second, we construct a vector database by extracting text from the PDFs, chunking the content into manageable segments, and encoding each chunk using sentence embeddings. These embeddings are stored in a FAISS index (Douze et al., 2024) structure that enables efficient similarity search during retrieval. Third, we generate prompts for our chatbot by combining questions from the UDA-QA dataset with the most relevant document chunks retrieved from the vector database. For each question, we encode the query text and perform a nearest neighbor search to identify the top matching chunks from the indexed PDF content. These retrieved chunks are then concatenated with the question to form the complete prompt that is sent to the language model. Finally, we collect the chatbot's generated answers and compare them against the ground truth answers provided in the dataset using automated evaluation metrics.

7.3 Evaluation Metrics

We assess the quality of generated answers using three complementary metrics that measure different aspects of textual similarity and semantic correctness. BLEU score evaluates the n-gram overlap between the predicted answer and the ground truth, providing a surface-level measure of how closely the generated text matches the reference. ROUGE score complements BLEU by focusing on recall-oriented metrics, specifically measuring how much of the reference answer content appears in the predicted answer. We report ROUGE-L, which considers the longest common subsequence between the two texts. BERTScore provides a semantic similarity measure by computing contextual embeddings for tokens in both the predicted and reference answers, then calculating precision, recall, and F1 scores based on the cosine similarity between these embeddings.

8 Test Design

8.1 Experimental Procedure

Our evaluation consists of two phases. The first phase assesses retrieval quality by measuring whether the vector database successfully retrieves relevant document chunks for each question. We construct FAISS indices (Douze et al., 2024) for each PDF document in UDA-QA (Hui et al., 2024) by extracting text, and encoding them with Sentence-BERT (Reimers and Gurevych, 2019). For each question, we encode the query and retrieve the top K nearest chunks using Euclidean distance, testing K values of 1, 5, 10, and 50. The second phase evaluates end-to-end question answering performance. We retrieve the top K chunks for each question and insert them into a simple prompt template with the context followed by the question. The chatbot generates answers using either Gemini 2.0 flash (Google DeepMind Team et al., 2023) or Gemini 2.5 flash (Comanici et al., 2025). We then compare generated answers against ground truth using BLEU score, ROUGE, and BERTScore.

8.2 Hyperparameter Configuration

We conduct ablation studies on two primary variables. First, we compare the performance of Gemini 2.0 Flash and Gemini 2.5 Flash as the answer generation models. Second, we vary the number of retrieved chunks K across values of 1, 5, 10, and 50 to determine the optimal amount of context for answer generation. All other parameters remain fixed, including the use of all-MiniLM-L6-v2 for sentence encoding, token chunk size, and token overlap.

8.3 Test Environment

All experiments run on a single AWS EC2 g5.2xlarge instance with 8 vCPUs, 32 GB RAM, and an NVIDIA A10G GPU. FAISS indices are stored locally on the same instance.

8.4 Reproducibility

All results can be reproduced using our evaluation script with command line arguments for dataset path, output directory, top K value, and model selection. Results are saved to a JSONL file containing question text, document category, retrieved chunks, generated answer, ground truth answer, and all computed metrics.

9 Deployment Model

9.1 Production Architecture

CyberLife AI is deployed using a hybrid cloud architecture that separates the frontend, backend, and model server into independent services. The frontend is hosted on Vercel, which provides serverless deployment and global edge caching through Cloudflare. The TypeScript codebase is compiled through Vite, bundled into optimized static assets, and served as a static site.

The backend runs on an AWS EC2 instance with Ubuntu 24.04. It is managed as a systemd service to ensure automatic restarts and consistent uptime. Nginx functions as a reverse proxy for SSL termination and routing traffic to the Python application. Backend configuration variables specify retrieval mode, embedding model, and other operational parameters, and the service handles both API requests and document indexing.

The model server is deployed on a dedicated GPU instance (g5.2xlarge) to avoid contention with the backend. Models are loaded once during startup and remain resident in GPU memory to reduce warmup time. Although the MVP uses a single model server, the architecture supports extension to multiple GPU nodes behind a load balancer for future scaling needs.

9.2 Monitoring and Logging

System behavior is tracked through structured JSON logs produced by the Python backend. Error logs include full stack traces to support rapid debugging. The system monitors key performance metrics, including request latency, embedding retrieval time, LLM generation time, and video generation time. These metrics guide performance tuning and identify potential bottlenecks.

10 Risks/Challenges

10.1 Technical Challenges

10.1.1 Model Server Bottleneck

Issue: SadTalker (Zhang et al., 2023) video generation takes 15-30 seconds per response, creating poor user experience.

Root Causes:

- 3DMM coefficient generation is sequential (no batching across users)

- Face rendering requires multiple GPU passes
- Video encoding with FFmpeg is CPU-bound

Mitigation Strategies:

- Implement request queue with batch processing
- Cache default avatar videos for common phrases
- Add “audio-only mode” toggle (already implemented)
- Explore faster avatar models (e.g., Wav2Lip (Prajwal et al., 2020))

10.1.2 Prompt Engineering Instability

Issue: LLM responses sometimes ignore retrieved context or hallucinate despite grounding.

Root Causes:

- Prompt template not optimized for RAG (Lewis et al., 2020)
- Retrieved chunks may be too long or verbose
- Model may rely on parametric knowledge over provided context

Mitigation Strategies:

- Add explicit instruction: ”ONLY use information from the context below”
- Implement citation: require model to quote source chunks

10.2 Data and Privacy Risks

10.2.1 User Data Security

Risk: User-uploaded documents and conversation history contain sensitive information.

Mitigation:

- All uploads stored with session-specific isolation
- No cross-user data leakage (verified via separate FAISS indices)
- Plan to implement encryption at rest for future production

10.2.2 Embedding Privacy

Risk: Embeddings may encode private information that could be reverse-engineered.

Mitigation:

- Store embeddings separately from raw text
- Implement differential privacy noise injection (future work)
- Consider on-device embedding generation for high-security use cases

11 Tools and Dependencies

11.1 Core Libraries

11.1.1 Backend (Python 3.9+)

Table 1: Python dependencies with rationale

Library	Version	Purpose
Flask	2.3.0	Web framework for REST API
flask-cors	4.0.0	CORS handling for frontend
google-generativeai	0.3.0	Gemini LLM API client
SpeechRecognition	3.10.0	Audio transcription
TTS (Coqui)	0.22.0	Text-to-speech synthesis
sentence-transformers	2.2.2	Embedding generation
faiss-cpu	1.7.4	Vector similarity search
PyPDF2	3.0.1	PDF text extraction
torch	2.0.1	Deep learning backend
numpy	1.24.3	Numerical operations
opencv-python	4.8.0	Image/video processing

Rationale for Key Choices:

- **Flask over FastAPI**: Simpler for MVP; sufficient performance for I/O-bound workload
- **FAISS over Pinecone**: Free, runs locally, faster for small datasets ($\leq$ 100k vectors)
- **Sentence-Transformers**: Easy-to-use wrapper around HuggingFace models
- **Coqui TTS**: Open-source, supports voice cloning, good voice quality

11.1.2 Model Server (Python 3.8)

SadTalker has strict dependency requirements:

- Python 3.8 (compatibility issues with 3.9+)
- PyTorch 1.12.1 with CUDA 11.6

11.1.3 Frontend (TypeScript/JavaScript)

Table 2: Frontend dependencies

Package	Version	Purpose
vite	5.0.0	Build tool and dev server
typescript	5.3.0	Type-safe development
tailwindcss	3.4.0	Utility-first CSS framework

11.2 External Services

- **Google Gemini API**: LLM generation (2.0/2.5 Flash model)
- **Google Speech Recognition API**: Audio transcription
- **SerpAPI (Optional)**: Web search fallback

11.3 Infrastructure

- **Vercel**: Frontend hosting and serverless functions
- **AWS EC2**: We use a g5.2xlarge instance to host our backend service.
- **FFmpeg**: Video/audio processing

11.4 Development Tools

- **Git**: Version control (GitHub)
- **VSCode**: Primary IDE
- **Postman**: API testing during development
- **Chrome DevTools**: Frontend debugging

- **pytest**: Python unit testing
- **ESLint + Prettier**: Code quality for TypeScript

11.5 Pre-existing Code

We build upon the following open-source projects:

- **SadTalker** (Zhang et al., 2023): Avatar animation system
  - GitHub: `https://github.com/OpenTalker/SadTalker`
- **Coqui TTS** (Casanova et al., 2022): Text-to-speech models
  - GitHub: `https://github.com/coqui-ai/TTS`

All other code is original development for this capstone project.

12 Results (Temporary)

We evaluated our PDF chunk retrieval system using 10 test questions from the FETA dataset (Hui et al., 2024), with Google Gemini 2.0 Flash (Google DeepMind Team et al., 2023) as the generation model. Table 3 shows performance across different top-K retrieval configurations.

Table 3: Performance metrics for varying top-K values (N=10 questions)

Metric	**Top-1**	**Top-5**	**Top-10**	**Top-50**
F1 Score	0.289	0.461	0.577	**0.656**
BLEU Score	0.060	0.194	0.275	**0.296**
ROUGE-1	0.323	0.476	0.583	**0.672**
ROUGE-2	0.160	0.306	0.400	**0.490**
ROUGE-L	0.262	0.440	0.517	**0.592**
BERTScore-F1	0.863	0.897	0.911	**0.916**

12.1 Key Findings

1. **Consistent improvement with increased context**: All metrics improve monotonically as K increases from 1 to 50, with F1 score increasing by 127% (0.289 $\rightarrow$ 0.656).
2. **Diminishing returns beyond K=10**: The largest gains occur between Top-1 and Top-5 (+60% F1), with marginal improvement from Top-10 to Top-50 (+14% F1).

3. **Strong semantic accuracy**: BERTScore F1 reaches 0.916 at Top-50, indicating semantically correct responses despite zero exact string matches.

13 Error Analysis

- **Zero Exact Match**: No predictions achieved exact string matching despite high semantic scores. Root cause: Gemini 2.0 (Google DeepMind Team et al., 2023) naturally paraphrases retrieved text rather than extracting verbatim answers. This indicates Exact Match is not suitable for evaluating generative RAG systems (Lewis et al., 2020).
- **BLEU-ROUGE Discrepancy**: BLEU remains low (max 0.296) while ROUGE-1 reaches 0.672 because BLEU penalizes word reordering heavily, whereas ROUGE-1 rewards content presence regardless of structure. The model captures correct information but uses different phrasing.
- **Noise at High K**: Top-50 occasionally includes irrelevant details from marginally related chunks.

14 Discussion

Our evaluation demonstrates that the dual-retrieval RAG (Lewis et al., 2020) pipeline successfully enables knowledge-grounded responses, with F1 scores improving from 0.289 to 0.656 as context increases. The high BERTScore (0.916) confirms semantic accuracy despite zero exact matches, validating that generative systems prioritize meaning over verbatim reproduction.

15 Lessons Learned and Reflections

At the beginning, we focused too much on the video response speed and quality and neglected more complete system construction. However, under the guidance and suggestions of mentor Adam and Professor Kemal, we got back on track and focused more on how to contribute value. We built a dual retriever system and used a vector database to store data and handle large amounts of user-specific files. We also added web search options and frontend deployment. We learned how to take an idea through all stages from conception and design to planning, implementation, verification, and presentation, while also integrating what we learned in MCDS into this capstone project (cloud computing, multimodal ML, database systems, etc.)

16 Future Work

16.1 Reducing Avatar Latency

Replace SadTalker with faster models like Wav2Lip (Prajwal et al., 2020), implement GPU batching and request queuing, and cache common phrases.

16.2 Scaling Architecture

Migrate to persistent vector databases like Pinecone, add PostgreSQL-backed user authentication, deploy multiple backend replicas with Redis caching, and support multi-agent collaboration.

16.3 Expanding Evaluation

Conduct systematic benchmarking with baselines, collect user feedback through studies, and perform A/B testing on retrieval modes.

These improvements directly address current shortcomings and advance the platform toward scalable, production-ready conversational agents.

17 Conclusion

CyberLife AI is a no-code platform that enables non-technical users to create personalized conversational agents with persistent memory and domain-specific knowledge. Our dual-retrieval RAG pipeline combines conversation history recall with document-based knowledge retrieval to produce agents that are both contextually coherent and factually grounded. Users can upload their own documents, define agent personalities, and integrate multimodal features like speech and animated avatars without writing code. Our evaluation on topic-specific datasets demonstrates that increasing retrieval context improves response quality, with F1 scores reaching 0.656 and BERTScore achieving 0.916. Future work will focus on improving retrieval accuracy, reducing avatar generation latency, and scaling the platform to support multi-agent collaboration and real-world deployment.

18 Acknowledgments

We would like to thank the TAs from both semesters, Shivang and Alice, as well as Mentor Adam, who provided significant help throughout the project design phase and communication

during subsequent direction changes.

19 Terminology, Definitions, Acronyms, and Abbreviations

This section defines the key technical terms, abbreviations, and system components used throughout the report. The goal is to ensure clarity and consistency for readers who may not be familiar with retrieval-augmented generation systems, memory architectures, or multimodal conversational agents. Table 4 summarizes the essential terminology.

Table 4: Key terminology and definitions used in CyberLife AI

Term	Definition
RAG (Retrieval-Augmented Generation)	A framework in which a language model augments its answers using external retrieved documents. In CyberLife, RAG combines conversation-memory retrieval with document retrieval to ensure grounded and coherent responses.
Dual-Retrieval Pipeline	CyberLife's architecture that uses two parallel retrievers: (1) a conversation memory retriever (recent turns or embedding-based long-term memory) and (2) a document retriever over user-uploaded PDFs, with optional fallback to web search.
Embedding	A dense vector representation of text used to compute semantic similarity. Generated using Sentence-BERT for both conversation history and document chunks.
Vector Database	A specialized database for storing embeddings and retrieving nearest neighbors using similarity search. CyberLife uses FAISS for this purpose.
FAISS	A high-performance library (Facebook AI Similarity Search) used for fast approximate nearest neighbor search over embeddings.

Continued on next page

Term	**Definition**
Chunk	A text segment produced by splitting longer documents into manageable sections for embedding and retrieval.
Session	A single user–agent interaction period. Each session has its own conversation history buffer and retrieval context.
Persona	A structured description of an agent's identity, behavior, tone, and traits. Prepended to prompts so the LLM responds consistently with the defined persona.
TTS (Text-to-Speech)	The subsystem that converts the agent's generated text into spoken audio. CyberLife uses Coqui TTS for high-quality synthesis with optional voice cloning.
STT (Speech-to-Text)	The subsystem converting user-recorded audio into text. Implemented using Google Speech Recognition API.
LLM (Large Language Model)	A foundational model such as Google Gemini 2.0 Flash used to generate responses conditioned on persona, conversation history, and retrieved knowledge.
SadTalker	A pretrained neural avatar animation model that generates talking-head videos synchronized to speech audio. Deployed on a separate GPU-backed model server.

These definitions serve as reference points for readers and will be used consistently across the remainder of the document.

References

Edresson Casanova, Julian Weber, Christopher D Shulby, Arnaldo Candido Junior, Eren Gölge, and Moacir A Ponti. 2022. Yourtts: Towards zero-shot multi-speaker tts and zero-shot voice conversion for everyone. In *International Conference on Machine Learning*, pages 2709–2720. PMLR.

Gheorghe Comanici, Eric Bieber, Mike Schaekermann, Ice Pasupat, Noveen Sachdeva, Inderjit Dhillon, Marcel Blistein, Ori Ram, Dan Zhang, Evan Rosen, and 1 others. 2025. Gemini 2.5: Pushing the frontier with advanced reasoning, multimodality, long context, and next generation agentic capabilities. *arXiv preprint arXiv:2507.06261*.

Matthijs Douze, Alexandr Guzhva, Chengqi Deng, Jeff Johnson, Gergely Szilvasy, Pierre-Emmanuel Mazaré, Maria Lomeli, Lucas Hosseini, and Hervé Jégou. 2024. The faiss library.

Google DeepMind Team, Rohan Anil, Sebastian Borgeaud, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, Katie Millican, and 1 others. 2023. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*.

Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. 2020. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning*, volume 119 of *Proceedings of Machine Learning Research*, pages 3929–3938. PMLR.

Yulong Hui, Yao Lu, and Huanchen Zhang. 2024. Uda: A benchmark suite for retrieval augmented generation in real-world document analysis. *Advances in Neural Information Processing Systems*, 37:67200–67217.

Bernhard Kerbl, Georgios Kopanas, Thomas Leimkühler, and George Drettakis. 2023. 3d gaussian splatting for real-time radiance field rendering. *ACM Trans. Graph.*, 42(4):139–1.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, and 1 others. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*, 33:9459–9474.

Charles Packer, Vivian Fang, Shishir_G Patil, Kevin Lin, Sarah Wooders, and Joseph_E Gonzalez. 2023. Memgpt: Towards llms as operating systems.

KR Prajwal, Rudrabha Mukhopadhyay, Vinay P Namboodiri, and CV Jawahar. 2020. A lip sync expert is all you need for speech to lip generation in the wild. In *Proceedings of the 28th ACM International Conference on Multimedia*, pages 484–492.

Nils Reimers and Iryna Gurevych. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Wenxuan Zhang, Xiaodong Cun, Xuan Wang, Yong Zhang, Xi Shen, Yu Guo, Ying Shan, and Fei Wang. 2023. Sadtalker: Learning realistic 3d motion coefficients for stylized audio-driven single image talking face animation. In *Proceedings of the IEEE/CVF conference on Computer Vision and Pattern Recognition*, pages 8652–8661.

A Appendix

We will keep this section for our detail experimental contents.

B Changes To Previous Deliverables

Over the course of the spring and fall terms, several substantial updates were made to the project compared to our original proposal, design document, and midterm report. These changes reflect both the evolving scope of the system and a shift toward a more research-driven and technically robust platform. Below, we summarize the major modifications and their motivations.

Shift from a Simple No-Code Builder to a Research-Oriented Dual-Retrieval System

In the original proposal, our focus centered primarily on creating an accessible no-code interface for building conversational agents. During the midterm phase, user feedback and technical exploration revealed the importance of grounding agents in both long-term memory and user-provided documents. As a result, the system pivoted toward a **dual-retrieval RAG pipeline**, incorporating:

- Embedding-based conversation memory retrieval,
- Document retrieval over user-uploaded PDFs, and
- A fallback web search mechanism.

This update required major changes to the system architecture sections in earlier deliverables.

Replacement of Simple History Buffer with Embedding-Based Memory

Earlier deliverables described a sliding-window memory design. In the final system, we replaced this with a **semantic memory subsystem** using Sentence-BERT embeddings and FAISS. The rationale, implementation details, and new retrieval modes were updated across the design document and system overview.

Addition of PDF Ingestion and User Knowledge Base

Previous documents assumed only text-based content. As the system matured, we added a full **PDF ingestion pipeline** including:

- PDF extraction,
- Chunking with overlap,
- Embedding generation,
- Vector store indexing.

Corresponding changes were made in the data design, system workflow diagrams, and requirements sections.

Significant Architecture Redesign

The midterm version described a single-server architecture. By the final report, we transitioned to a **three-tier microservices architecture**:

- Vercel TypeScript frontend,
- Python Flask backend for retrieval and conversation logic,
- GPU-enabled model server for avatar generation.

The deployment model, system overview, and interface descriptions were updated accordingly.