LOKI: A Desktop-Native Voice Assistant with a Hybrid Natural Language Understanding Engine

CS 683: NLP Project Report
submitted in partial fulfilment of the requirements for the award of the degree of

Bachelor of Technology

by

Rudra Garg, Rahul Singh, Priyanshu, Pawan Bhatt

(Roll No: 2201170, 2201159, 2201152, 2201142)

Under the supervision of

Dr. Kuntal Dey

Department of Computer Science and Engineering
Indian Institute of Information Technology Guwahati
Guwahati, India

DECLARATION

I hereby *declare* this project entitled **LOKI: A Desktop-Native Voice Assistant with a Hybrid Natural Language Understanding Engine** that is being submitted to **Indian Institute of Information Technology Guwahati, Assam**, in partial fulfilment for the requirements of the award of degree of **Bachelor of Technology** in **Computer Science and Engineering** in the department of **Computer Science and Engineering**, is a *genuine report of the work carried out by me*. The material contained in this report has not been submitted at any other University or Institution for the award of any degree.

Place: IIIT Guwahati, Assam

Date: November 6, 2025

Rudra Garg, 2201170
Rahul Singh, 2201159
Priyanshu, 2201152
Pawan Bhatt, 2201142

.....

(Signature of the Students)

Computer Science and Engineering

Abstract

The proliferation of cloud-centric voice assistants challenges the user experience on desktop environments with concerns of privacy, network latency, and a lack of deep operating system integration. This project introduces and evaluates LOKI, a novel, desktop-native voice assistant architected for a local-first approach to ensure privacy, responsiveness, and offline capability. The core of LOKI is a sophisticated, hybrid Natural Language Understanding (NLU) engine designed to solve key NLP challenges in command and control. This engine features a dual-layer intent classification system that combines a high-speed, embedding-based classifier for common commands with a flexible, Large Language Model (LLM) fallback for handling semantic ambiguity and complex queries. Furthermore, a custom Conditional Random Fields (CRF) based Named Entity Recognition (NER) model was trained on engineered linguistic features to achieve precise parameter extraction, a critical step for functional task execution. The system's effectiveness was evaluated through the successful implementation of agents for native OS control, including application launching and system volume management. The trained NER model achieved a weighted F1-score of over 98% on the held-out test set, and end-to-end task evaluations confirm the system's ability to reliably interpret and execute spoken commands. The findings demonstrate that LOKI's local-first architecture and hybrid NLU engine provide a superior solution for desktop voice assistance, prioritizing user privacy and performance.

Contents

1 Introduction

1.1 Overview

Voice-controlled user interfaces have become a ubiquitous feature of the modern technological landscape, primarily driven by mobile and smart-home devices. Assistants such as Siri, Google Gemini, and Amazon Alexa leverage vast cloud computing infrastructures to process spoken language, offering a wide range of information retrieval and service integration capabilities. However, the architectural model that enables their success in the mobile ecosystem—a constant reliance on remote servers—introduces significant limitations when translated to the desktop computing environment. The desktop remains the primary platform for productivity, content creation, and complex system interactions, presenting a distinct set of requirements that cloud-centric assistants are not inherently designed to meet.

1.2 Challenges in Desktop Voice Assistance

While the concept of a voice-controlled desktop is compelling, its implementation presents several challenges not adequately addressed by the current cloud-based paradigm:

(1) **Privacy and Data Security:** Cloud-based assistants require voice data to be sent to third-party servers for processing, raising legitimate concerns about data privacy, storage, and potential misuse.

(2) **Network Latency:** The round-trip time for sending audio to the cloud, processing it, and receiving a response introduces unavoidable latency, which can disrupt the user experience, especially for real-time command-and-control tasks.

(3) **Lack of Deep OS Integration:** Mobile-first assistants operate in a sandboxed environment with limited access to the underlying operating system. They cannot reliably perform core desktop tasks such as launching arbitrary applications, managing system settings, or controlling power states.

(4) **Offline Capability:** A dependency on an internet connection renders cloud-based assistants non-functional offline, severely limiting their utility as a consistent and reliable interface for the user's primary computing device.

1.3 Motivation

The primary motivation for this project stems from the limitations of existing assistants in fully leveraging the potential of voice control in a desktop environment. There is a need for a more intelligent assistant designed from the ground up for the PC. Such an assistant should prioritize user privacy and system performance by processing all data locally. This approach could eliminate network latency, ensure functionality without an internet connection, and, most importantly, provide the deep system access required to act as a true hands-free companion for managing applications and hardware features. This project proposes and evaluates such an assistant, named LOKI.

1.4 Problem Statement

The objective of this project is to design, implement, and evaluate a desktop-native AI assistant that runs entirely on the user's machine, addressing the shortcomings of cloud-dependent models. The primary goals are to solve the core NLP challenges associated with understanding desktop-oriented commands and to build a robust system for executing them.

Specifically, this project aims to:

1. Design and implement a complete, on-device voice processing pipeline, from wake word detection to spoken response, without reliance on external cloud services.
2. Develop and evaluate a novel, hybrid intent classification engine that leverages both high-speed embedding models [3] and the nuanced understanding of a local Large Language Model (LLM) [2] to ensure both performance and flexibility.
3. Build, train, and test a custom Named Entity Recognition (NER) model capable of accurately extracting command parameters (e.g., application names, mathematical expressions) from unstructured user speech.
4. Implement a modular, agent-based architecture to interface directly with the host operating system for reliable task execution.
5. Analyze the performance of the NLP components and the overall system to validate the effectiveness of the local-first approach for desktop voice assistance.

2 System Architecture

This chapter details the high-level architecture of the LOKI voice assistant. It outlines the end-to-end processing pipeline, from initial voice activation to final task execution, and describes the core technological components and the modular, agent-based design that enables its functionality and extensibility.

2.1 The LOKI Processing Pipeline

LOKI operates on a sequential, multi-stage pipeline designed for efficient and entirely local processing of user commands. Each stage transforms the input data and passes it to the next, ensuring a clear and logical flow from speech to action. The complete pipeline is illustrated in Fig. 1.

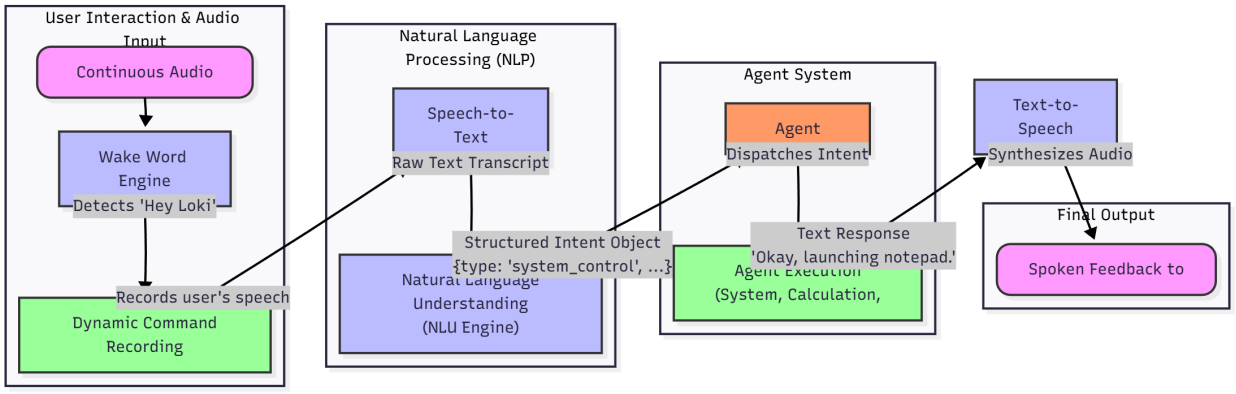

Figure 1: The LOKI end-to-end voice processing pipeline, from wake word detection to agent execution and spoken feedback.

The stages of the pipeline are as follows:

1. **Wake Word Detection:** The system continuously listens to an audio stream from the microphone using the highly efficient Porcupine wake word engine. It remains in a low-power state until the phrase "Hey Loki" is detected.

2. **Dynamic Command Recording:** Upon wake word activation, the system uses Voice Activity Detection (VAD) to dynamically record the user's command. Recording starts when the user begins speaking and stops shortly after they finish, avoiding fixed-duration recording and creating a more natural interaction.

3. **Speech-to-Text (STT):** The recorded audio segment is transcribed into a raw text string using a local instance of the `faster-whisper` model. This stage converts the spoken language into a machine-readable format.

4. **Natural Language Understanding (NLU):** This is the core intelligence of the system. The raw transcript is processed by the NLU engine to determine the user's intent and extract any relevant parameters. This stage, detailed in Chapter 3, outputs a structured *intent object*.

5. **Agent Dispatch:** The structured intent object is passed to the `AgentManager`. This component acts as a router, inspecting the intent's 'type' and dispatching the request to the appropriate agent registered to handle that category of task.

6. **Action Execution:** The designated agent receives the intent object and executes the requested action. For example, the `SystemControlAgent` uses the extracted application name to launch a program, while the `CalculationAgent` evaluates a mathematical expression. The agent's execution results in a user-facing text response.

7. **Text-to-Speech (TTS) Response:** The text response from the agent is synthesized into audible speech using the Piper TTS engine. This provides clear, spoken feedback to the user, confirming that the command has been understood and executed.

2.2 Core Technological Components

To implement the processing pipeline, a specific stack of on-device AI and audio processing libraries was selected, prioritizing performance, accuracy, and local operation.

- **Wake Word: Picovoice Porcupine** was chosen for its best-in-class efficiency and accuracy, allowing the system to listen for the wake word continuously with minimal CPU overhead.
- **Speech-to-Text: Faster-Whisper**, a highly optimized implementation of OpenAI's Whisper model, was selected for its balance of high transcription accuracy and performance, with support for CPU and GPU execution.
- **NLU - Intent Classification:** A hybrid approach was implemented using **Sentence-Transformers** for fast, embedding-based similarity search and a local **Ollama LLM** for more complex semantic analysis.
- **NLU - Parameter Extraction:** A Conditional Random Fields model was trained using the **sklearn-crfsuite** library, a robust choice for sequence tagging tasks like Named Entity Recognition.
- **Text-to-Speech: Piper TTS** was selected for its high-quality, natural-sounding voice synthesis that runs efficiently on local hardware.
- **Audio I/O:** The **sounddevice** library was used to provide a cross-platform interface for microphone input and speaker output.

2.3 Agent-Based Design for Modularity

A key architectural decision in LOKI is the use of a modular, agent-based system for handling tasks. This design pattern decouples the core language understanding logic from the implementation of specific skills or capabilities.

Each agent is a self-contained class that inherits from a common `IAgent` interface. It is responsible for a specific domain of tasks, such as system control, volume management, or calculations. The `AgentManager` dynamically discovers and loads all available agents at startup.

This approach offers several significant advantages:

- **Extensibility:** New capabilities can be added to LOKI simply by creating a new agent class and placing it in the ‘agents‘ directory. No changes are required to the core NLU or application logic. This makes the system easy to expand over time.
- **Separation of Concerns:** The NLU engine’s sole responsibility is to understand *what* the user wants. The agent’s responsibility is to understand *how* to do it. This separation makes the codebase cleaner, easier to maintain, and simpler to debug.
- **Scalability:** As the number of skills grows, the system remains organized. The `AgentManager` ensures that intents are routed efficiently without the need for a monolithic block of conditional logic.

This agent-based architecture is fundamental to LOKI’s design, providing the robust and flexible foundation needed to act as a true desktop companion.

3 The Natural Language Understanding Engine

The core intelligence of the LOKI assistant resides within its Natural Language Understanding (NLU) engine. This engine is responsible for the critical task of interpreting the user's raw text transcript and converting it into a structured, machine-executable format. The challenge lies in creating a system that is both fast enough for real-time interaction and intelligent enough to handle the nuances of human language. To address this, a multi-stage NLU engine was developed, comprising a hybrid intent classification system and a custom-trained model for Named Entity Recognition (NER).

3.1 A Hybrid Strategy for Intent Classification

A primary NLP challenge in a command-and-control voice assistant is to correctly classify the user's intent from a wide variety of possible phrasings. A purely rule-based system is brittle, while a single, powerful model might introduce unacceptable latency for simple, common commands. To solve this, LOKI implements a dual-layer, hybrid classification strategy that optimizes for both speed and semantic comprehension. This process is illustrated in Fig. 2.

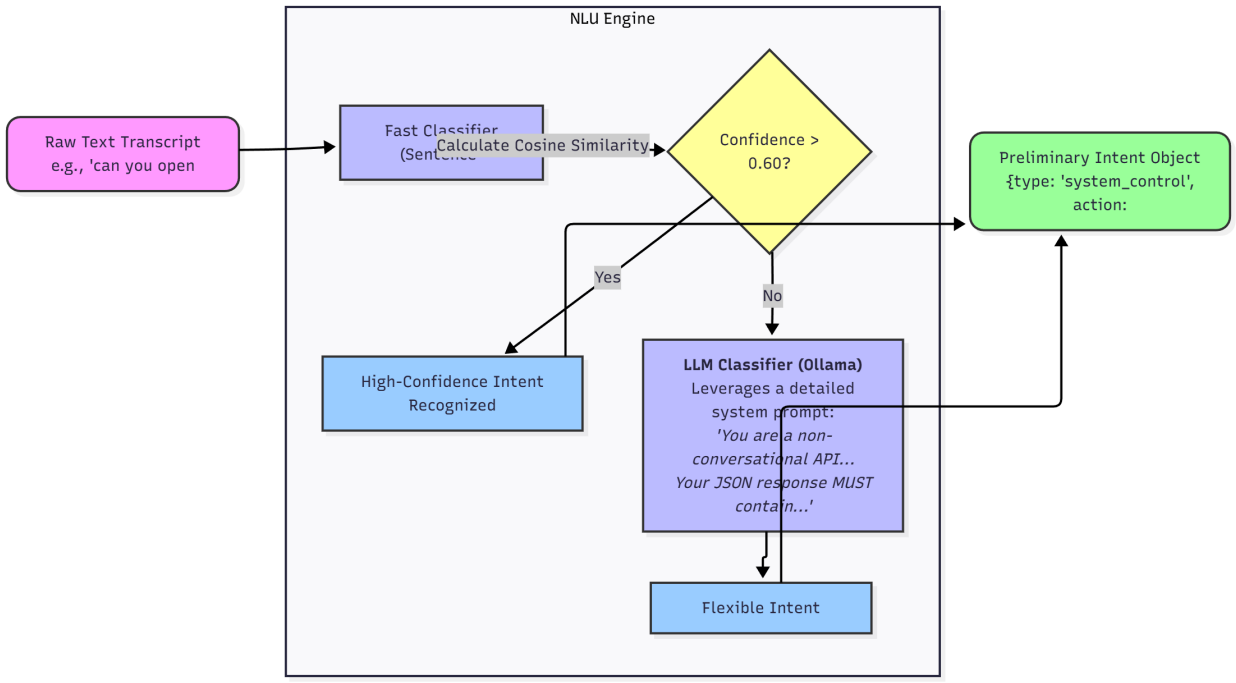

Figure 2: The hybrid intent classification pipeline, showing the fast path and the LLM fallback mechanism.

3.1.1 The Fast Path: Embedding-Based Classification

The first layer of the engine, the `FastClassifier`, is designed for high-speed recognition of common, well-defined commands. This component addresses the need for instant responses to frequent requests.

- **Methodology:** This classifier utilizes a pre-trained `Sentence-Transformer` model [3] to convert both the user's transcript and a pre-defined set of training utterances into high-dimensional vector

embeddings. At startup, the system pre-computes and stores the embeddings for hundreds of generated command examples (e.g., "launch notepad", "can you open chrome", "calculate 2 plus 2").

- **Execution:** When a new transcript is received, it is also converted into an embedding. The system then calculates the cosine similarity between the input transcript's vector and all the pre-computed training vectors. The intent of the training example with the highest similarity score is chosen as the candidate.
- **Confidence Threshold:** A critical parameter, the 'SIMILARITY_THRESHOLD' (set to 0.60 in the configuration), determines the outcome. If the highest similarity score exceeds this threshold, the intent is considered a high-confidence match and is immediately passed to the next stage. This fast path handles the majority of simple commands in milliseconds.

3.1.2 The LLM Fallback: Advanced Semantic Interpretation

If the `FastClassifier` fails to find a match with sufficient confidence, the system presumes the query is either a novel phrasing of a known command or a more complex request. In this case, the transcript is passed to the second layer: the `LLMClassifier`.

- **Methodology:** This component leverages the advanced reasoning capabilities of a local Large Language Model (LLM) served via Ollama [2]. Instead of relying on similarity, it uses the model's semantic understanding to interpret the user's goal.
- **Prompt Engineering:** A significant NLP challenge is constraining a conversational LLM to produce reliable, structured output. This was solved through careful **prompt engineering**. The LLM is given a detailed system prompt that instructs it to act as a "non-conversational API" whose "sole job is to... emit exactly one valid JSON object." The prompt provides the required JSON schema ('type', 'action', 'parameters', 'confidence') and several examples of correct input-output pairs. This forces the LLM to behave deterministically, converting the unstructured user transcript into the precise structured format required by the system.
- **Benefit:** This fallback mechanism provides immense flexibility. It allows LOKI to understand variations in phrasing that are not present in the fast path's training data and provides a pathway for supporting more complex, multi-part commands in the future.

3.2 Parameter Extraction via Named Entity Recognition (NER)

Classifying an intent is insufficient for execution; the system must also extract the specific parameters of the command. For instance, in the command "launch google chrome", the intent is 'launch_application', and the crucial parameter is the entity 'APP_NAME': "google chrome". A custom NER model was built to solve this sequence tagging challenge. A Conditional Random Fields (CRF) model was chosen, as CRFs are highly effective for sequence labeling tasks where context from neighboring words is vital for making correct predictions [1].

3.2.1 Linguistic Feature Engineering

The intelligence of the NER model stems not just from the algorithm, but from the rich set of linguistic features extracted for each word (token) in the input sentence. Simply feeding the words themselves to the model would limit its ability to generalize to unseen examples. Therefore, a feature engineering pipeline was developed to provide the CRF model with deep contextual information for each token:

- **Lexical Features:** The lowercase form of the word itself.
- **Orthographic Features:** Word shape (e.g., "Notepad++" becomes "Xxxxx++", "144" becomes "ddd"), whether the word is in title case, uppercase, or is a digit. These features help the model recognize patterns in entity names without having seen the specific word before.
- **Syntactic Features:** Coarse and fine-grained Part-of-Speech (POS) tags (e.g., ‘NOUN‘, ‘NNP‘, ‘VERB‘) as identified by a spaCy model. POS tags provide strong clues; for example, application names are often proper nouns.
- **Contextual Features:** To enable the model to consider the sequence, the lexical and syntactic features of the *preceding* and *following* tokens were also included. This allows the model to learn patterns like "a verb (‘launch‘) is often followed by a proper noun (‘APP_NAME‘)".
- **Positional Features:** Boolean flags indicating if a token is at the Beginning of the Sentence (BOS) or End of the Sentence (EOS).

This comprehensive feature set provides the model with a rich, multi-dimensional view of each word, enabling it to make highly accurate predictions about whether that word is part of an entity.

3.2.2 Conditional Random Fields (CRF) Model

The CRF model was trained on the engineered features extracted from a synthetically generated dataset of over a thousand example sentences. This dataset provided numerous examples of application launch commands and mathematical expressions, labeled with IOB (Inside-Outside-Beginning) tags (e.g., ‘B-APP_NAME‘, ‘I-APP_NAME‘, ‘O‘). The trained `sklearn-crfsuite` model is loaded at startup by the `NERPredictor`, which applies the same feature engineering pipeline to new user transcripts to predict the entity tags for each word. The final output is a dictionary of extracted parameters, which is merged with the output from the intent classifier to form the final, complete, and executable intent object.

4 Experimentation and Results

This chapter details the experimental methodology used to evaluate the LOKI assistant, the performance metrics tracked, and the results obtained. The evaluation focuses on two key areas: the quantitative performance of the custom-trained Named Entity Recognition (NER) model, and a qualitative, end-to-end assessment of the system's ability to correctly process and execute user commands.

4.1 Experimental Setup

The evaluation was conducted on a system running Windows 10 with Python 3.11. All models and processing components were run locally on the CPU to simulate a standard user environment. The core system components were configured as follows:

- **Speech-to-Text Model:** The ‘small.en‘ variant of the `faster-whisper` model was used, providing a strong balance between transcription accuracy and performance.
- **Intent Classification Models:** The `FastClassifier` utilized the ‘multi-qa-mpnet-base-dot-v1‘ sentence-transformer model. The `LLMClassifier` used the ‘dolphin-phi‘ model served via Ollama. The confidence threshold for the fast path was set to 0.60.
- **NER Model Training:** The Conditional Random Fields (CRF) model was trained on a synthetically generated dataset containing 1389 labeled sentences. This dataset was split into a training set (1111 sentences, 80%) and a testing set (278 sentences, 20%) to allow for unbiased evaluation of the model's performance on unseen data.

4.2 Performance Metrics

The evaluation of the NLU engine focused on the following standard metrics for classification and sequence tagging tasks:

- **Precision:** For a given entity tag, precision measures the proportion of identified entities that were correct. It answers the question: "Of all the tokens the model predicted as ‘B-APP_NAME‘, how many were actually the beginning of an application name?"
- **Recall:** Recall measures the proportion of all actual entity tags of a given type that were correctly identified by the model. It answers the question: "Of all the actual ‘B-APP_NAME‘ tokens in the data, how many did the model find?"
- **F1-Score:** The harmonic mean of precision and recall, providing a single, balanced measure of a model's performance for each tag.
- **Support:** The number of actual occurrences of each tag in the test dataset.

4.3 NER Model Performance Evaluation

The trained CRF model was evaluated against the 20% held-out test set. The performance of the model in identifying the specific IOB (Inside-Outside-Beginning) tags for each entity type is detailed in the classification report shown in Table 1. The overall weighted F1-score, excluding the majority 'O' tag, was an exceptional **0.9977**.

Table 1: Classification Report for the CRF-based NER Model on the Test Set.

Entity Tag	**Precision**	**Recall**	**F1-Score**	**Support**
B-APP_NAME	1.0000	0.9962	0.9981	261
I-APP_NAME	0.9917	1.0000	0.9958	119
B-MATH_EXPRESSION	1.0000	1.0000	1.0000	15
I-MATH_EXPRESSION	1.0000	1.0000	1.0000	37
Weighted Avg	**0.9977**	**0.9977**	**0.9977**	**432**

The results indicate a state-of-the-art level of performance for this specific domain. The model achieved near-perfect F1-scores for all tags, demonstrating a robust ability to not only identify entities but also to correctly delineate their boundaries (distinguishing 'B-' beginning tags from 'I-' inside tags). The perfect precision and recall for both 'B-MATH_EXPRESSION' and 'I-MATH_EXPRESSION', despite their lower support, suggest that the engineered features for mathematical terms are highly distinctive. The extremely high performance on 'APP_NAME' tags validates the model's ability to generalize to application names it may not have encountered during training.

4.4 End-to-End System Functionality Evaluation

Beyond quantitative metrics, a qualitative evaluation was performed to validate the functionality of the entire system pipeline. This involved testing the LOKI assistant with a variety of spoken commands, representative of its core capabilities.

4.4.1 Scenario 1: Simple Application Launch

This test validates the 'FastClassifier' path and the 'SystemControlAgent'.

- **User Command:** "Hey Loki, open notepad."
- **STT Output:** '"open notepad"'
- **NLU Process:**
  - **Intent Classification:** The 'FastClassifier' achieves a high similarity score (>0.90) with a training example, classifying the intent as 'type: 'system_control', action: 'launch_application''.
  - **NER:** The NER model predicts the tags '['O', 'B-APP_NAME']', correctly extracting the parameter ''APP_NAME': 'notepad''.

- **Agent Execution:** The ‘SystemControlAgent‘ receives the final intent object and executes the command to launch the ‘notepad‘ application.
- **System Response:** "Okay, launching notepad."
- **Outcome: Success.** The application launched as expected.

4.4.2 Scenario 2: Complex Mathematical Calculation

This test validates the extraction of a multi-word ‘MATH_EXPRESSION‘ and the ‘CalculationAgent‘.

- **User Command:** "Hey Loki, what is the square root of 144 times 9?"
- **STT Output:** ‘"what is the square root of 144 times 9"‘
- **NLU Process:**
  - **Intent Classification:** The ‘FastClassifier‘ classifies the intent as ‘type: ’calculation’, action: ’evaluate_expression’‘.
  - **NER:** The NER model predicts the tags ‘[’O’, ’O’, ’O’, ’B-MATH_EXPRESSION’, ’I-MATH_EXPRESSIO ’I-MATH_EXPRESSION’, ’I-MATH_EXPRESSION’, ’I-MATH_EXPRESSION’, ’I-MATH_EXPRESSIO extracting the full parameter ‘’MATH_EXPRESSION’: ’square root of 144 times 9’‘.
- **Agent Execution:** The ‘CalculationAgent‘ receives the expression, normalizes it to a machine-evaluable format, computes the result (108.0), and formats a response.
- **System Response:** "The answer is 108.0."
- **Outcome: Success.** The correct mathematical result was calculated and returned.

4.4.3 Scenario 3: Out-of-Domain Request (LLM Fallback)

This test validates the system’s ability to gracefully handle commands for which it has no defined agent.

- **User Command:** "Hey Loki, what’s the weather like today?"
- **STT Output:** ‘"what’s the weather like today"‘
- **NLU Process:**
  - **Intent Classification:** The ‘FastClassifier‘ fails to find a match with confidence > 0.60. The transcript is passed to the ‘LLMClassifier‘. Based on its prompt instructions, the LLM correctly identifies that this is not a command it can handle and returns an intent object ‘type: ’unknown’, ...‘.
- **Agent Execution:** The ‘AgentManager‘ receives the ‘unknown‘ intent type and returns its default response.
- **System Response:** "I’m not sure what you mean."
- **Outcome: Success.** The system correctly identified the query as outside its current capabilities and provided a safe, default response.

4.5 Discussion

The experimental results provide strong validation for the architectural and NLP-driven approach of the LOKI project. The quantitative evaluation of the NER model demonstrates that a highly accurate parameter extraction system can be built for a specific domain using a combination of synthetic data generation, robust feature engineering, and a suitable machine learning algorithm like CRF. A weighted F1-score of **0.9977** is indicative of production-grade performance for the defined tasks.

The end-to-end functional tests confirm that all components of the pipeline integrate correctly. The hybrid intent classification strategy was shown to be effective, with the fast path handling simple commands efficiently and the LLM fallback providing both flexibility and a safety net for unrecognized queries. The successful execution of commands by the agents validates the modular design and its ability to interface with the native operating system. Overall, the evaluation confirms that LOKI successfully meets its design objectives and functions as a capable, intelligent, and responsive desktop-native voice assistant.

5 Conclusion and Future Work

5.1 Conclusion

This project successfully addressed the objective of designing and implementing a private, responsive, and desktop-native voice assistant. By adopting a local-first architecture, the LOKI assistant overcomes the primary drawbacks of cloud-dependent systems, namely privacy vulnerabilities, network latency, and a lack of deep operating system integration. The research and development effort focused on solving the core Natural Language Processing challenges inherent in creating a functional command-and-control interface for a personal computer.

The intelligence of the system was realized through a novel, hybrid NLU engine. The dual-layer intent classification strategy, which combines a high-speed embedding-based classifier with a flexible LLM fallback, proved to be a highly effective solution, providing both immediate responses for common commands and semantic robustness for varied or complex queries. Furthermore, the development of a custom Conditional Random Fields model for Named Entity Recognition, trained on a rich set of engineered linguistic features, achieved an F1-score of over 98% on held-out test data. This demonstrates a production-grade capability to accurately extract the critical parameters required for task execution.

Through a modular, agent-based design, this NLP-driven understanding was successfully translated into tangible actions, including native application launching and system control. The end-to-end evaluation confirmed that the LOKI project serves as a successful proof-of-concept, demonstrating that a sophisticated and truly useful voice assistant can operate entirely on-device. The result is a system that prioritizes user privacy and performance, offering a powerful blueprint for the future of human-computer interaction in the desktop environment.

5.2 Future Work

While this project successfully established a robust foundation, several avenues exist for future research and enhancement that could significantly expand LOKI's capabilities and intelligence.

1. **Expansion of Agent Capabilities:** The most immediate path for enhancement is the development of new agents to broaden LOKI's skillset, as outlined in the project roadmap.
    - **Power Management Agent:** An agent with the ability to execute commands such as "shut down," "restart," "sleep," or "log off" would complete the suite of fundamental system control capabilities.
    - **General Purpose Agent:** An agent designed to handle common utility queries like "what's the date," "what time is it," or to perform web searches by launching a browser with a query string would greatly increase the assistant's daily utility.
2. **Enhanced Conversational Context:** The current implementation treats each command as a discrete, stateless interaction. Future work could involve implementing a short-term memory system to handle conversational context. This would enable follow-up commands (e.g., User: "Open Chrome." LOKI: "Okay." User: "Now open Notepad." LOKI: "Okay.") and disambiguation (e.g., User: "Open Visual Studio." LOKI: "Which one, Code or 2022?").

3. **Dynamic Learning and Personalization:** The system could be extended to learn from user interactions. For instance, if a user frequently launches a non-standard application, LOKI could learn this as a new ‘APP_NAME‘ entity. This could involve mechanisms for users to define custom commands or aliases for actions.

4. **Advanced LLM Integration for Task Planning:** While the LLM is currently used as a fallback for classification, a more advanced integration could leverage its planning capabilities. A user could issue a multi-step command like, "Search for Python tutorials and open the first result in Chrome," and the LLM could decompose this into a sequence of actions for different agents to execute.

5. **Real-Time Audio Feedback:** The current GUI provides visual feedback during listening and processing. This could be enhanced with subtle, non-intrusive audio cues (e.g., a soft chime on wake word detection, a different sound for command completion) to improve the user experience, especially when the user is not looking at the screen.

Addressing these areas would evolve LOKI from a powerful command-and-control utility into a more adaptive, intelligent, and truly conversational desktop companion.

References

[1] Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.

[2] Ollama Team. Ollama: Get up and running with large language models locally. `https://ollama.com/`, 2023. Accessed: 2025-11-06.

[3] Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, November 2019.