

## Design and Implementation of a Conversational AI Chatbot for Security Insights in Cisco Cyber Vision

**Auteur :** Hoorelbeke, Jordi

**Promoteur(s) :** Geurts, Pierre

**Faculté :** Faculté des Sciences appliquées

**Diplôme :** Master en sciences informatiques, à finalité spécialisée en "computer systems security"

**Année académique :** 2024-2025

**URI/URL :** <http://hdl.handle.net/2268.2/23371>

---

### Avertissement à l'attention des usagers :

*Tous les documents placés en accès ouvert sur le site le site MatheO sont protégés par le droit d'auteur. Conformément aux principes énoncés par la "Budapest Open Access Initiative"(BOAI, 2002), l'utilisateur du site peut lire, télécharger, copier, transmettre, imprimer, chercher ou faire un lien vers le texte intégral de ces documents, les disséquer pour les indexer, s'en servir de données pour un logiciel, ou s'en servir à toute autre fin légale (ou prévue par la réglementation relative au droit d'auteur). Toute utilisation du document à des fins commerciales est strictement interdite.*

*Par ailleurs, l'utilisateur s'engage à respecter les droits moraux de l'auteur, principalement le droit à l'intégrité de l'oeuvre et le droit de paternité et ce dans toute utilisation que l'utilisateur entreprend. Ainsi, à titre d'exemple, lorsqu'il reproduira un document par extrait ou dans son intégralité, l'utilisateur citera de manière complète les sources telles que mentionnées ci-dessus. Toute utilisation non explicitement autorisée ci-avant (telle que par exemple, la modification du document ou son résumé) nécessite l'autorisation préalable et expresse des auteurs ou de leurs ayants droit.*

---



UNIVERSITY OF LIÈGE  
SCHOOL OF ENGINEERING AND COMPUTER SCIENCE

---

# Design and Implementation of a Conversational AI Chatbot for Security Insights in Cisco Cyber Vision

---

Enabling Natural Language Interaction Through Cisco Webex  
Integration

*Author*

Jordi HOORELBEKE

*Supervisors*

Prof. Pierre GEURTS  
Emmanuel TYCHON

Master's thesis completed in order to obtain the degree of Master of  
Science in Computer Science

Academic year 2024-2025

# Abstract

The concept of natural human-computer interaction has long inspired both science fiction and early AI research. Thanks to advances in conversational AI, this vision is becoming increasingly practical, enabling more accessible and intuitive ways to interact with complex systems. Building on this progress, this thesis presents a chatbot designed to interface with Cisco Cyber Vision (CCV), a security platform for Operational Technology (OT) environments.

The chatbot empowers users to retrieve security insights through natural, intuitive messages. Integrated into Cisco Webex, it offers a familiar, text-based environment for questions like “What activities occurred between devices  $X$  and  $Y$  this week?”, “Explain vulnerability  $Z$ ,” or for receiving timely updates on the industrial network’s security posture. Its purpose is to streamline information access and reduce reliance on rigid web user interfaces.

This work’s key contributions include an integration environment assessment encompassing Cisco Webex, CCV, and the OT environment; the definition of conversational requirements for specifically tailored use cases; and the design and implementation of the chatbot’s architecture, primarily focusing on its conversational AI logic as the central contribution. To this end, we provide a detailed comparison of conventional machine learning models and modern Large Language Models (LLMs) for natural language understanding, detail how the chatbot adapts to non-standard language and typographical errors, and examine how generative AI and prompt engineering foster richer, more explanatory responses, enabling innovative use cases.

This work conclusively highlights the feasibility and benefits of augmenting CCV with a natural language interface via Webex. It also acknowledges and addresses inherent limitations introduced by specific design choices and the constraints of the Webex integration model.

## Acknowledgments

I would like to express my gratitude to my two supervisors, Pierre Geurts and Emmanuel Tychon, for their continuous support and invaluable guidance, which were instrumental in the success of my work. I am also thankful to Sophia Gurenath, a member of the Cisco Cyber Vision team at Cisco Systems US, for her essential feedback during the scope definition phases. Finally, I extend my sincere appreciation to the University of Liège and Cisco Systems for offering me the opportunity to work on such an enriching and intellectually stimulating project.

This project has been a significant milestone in my academic journey. Beyond the technical achievements, it has greatly contributed to my professional and personal growth, teaching me resilience, critical thinking, and the ability to overcome challenges inherent to complex projects.



## Disclaimer

The content of this thesis was written independently by the author and was not generated by any generative AI tool. Tools such as the grammar checker of [QuillBot](#) and [ChatGPT](#) were used solely to improve language clarity, grammar, and flow, without contributing to the substantive content or ideas presented.

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Origins of Chatbots: Early Milestones in Tech and Culture . . . . .	1
1.2	Objectives of the Thesis . . . . .	1
1.3	Structure of the Thesis . . . . .	3
<b>2</b>	<b>Integration Environment</b>	<b>5</b>
2.1	Cisco Systems . . . . .	5
2.2	Operational Technology (OT) and Industrial Control Systems (ICS) . . . .	5
2.2.1	Level 0: Physical Process . . . . .	6
2.2.2	Level 1: Basic Control . . . . .	6
2.2.3	Level 2: Supervisory Control . . . . .	6
2.2.4	Level 3: Manufacturing Operations . . . . .	6
2.2.5	Beyond Level 3 . . . . .	6
2.3	Cisco Cyber Vision (CCV) . . . . .	7
2.3.1	CCV Sensors . . . . .	7
2.3.2	CCV Center . . . . .	8
2.3.3	Concepts and Functions of the CCV Center . . . . .	8
2.3.4	Limitations of the CCV Center Web User Interface . . . . .	13
2.3.5	CCV Center Application Programming Interface (API) . . . . .	14
2.4	Cisco Webex . . . . .	14
2.4.1	Bot Integration in Webex . . . . .	14
<b>3</b>	<b>Conversational Requirements</b>	<b>16</b>
3.1	Conversational Artificial Intelligence (AI) . . . . .	16
3.2	Natural Language Understanding (NLU) . . . . .	16
3.2.1	Intent Classification (IC) . . . . .	17
3.2.2	Entity Recognition (ER) . . . . .	17
3.3	Natural Language Generation (NLG) . . . . .	17
<b>4</b>	<b>Functional Scope of the Chatbot</b>	<b>18</b>
4.1	Use-Case Definition . . . . .	18
<b>5</b>	<b>System Architecture Overview</b>	<b>20</b>
5.1	Chatbot Server . . . . .	22
5.2	Authenticator . . . . .	22
5.3	Interpreter . . . . .	22
5.4	Handler . . . . .	22
5.5	Assistant . . . . .	22

<b>6</b>	<b>Chatbot Server</b>	<b>23</b>
6.1	Server Initialization . . . . .	23
6.1.1	Web Server Configuration . . . . .	23
6.1.2	Webhook Configuration . . . . .	24
6.2	Message Reception . . . . .	24
6.3	Response Delivery . . . . .	25
<b>7</b>	<b>Authenticator</b>	<b>26</b>
7.1	CCV Center Authentication Mechanism . . . . .	26
7.2	Authentication Persistence . . . . .	27
7.3	Credential Collection and Confidentiality . . . . .	28
7.3.1	Adaptive Card . . . . .	28
7.4	Integration and Implementation Details . . . . .	28
7.4.1	Credential Validation . . . . .	29
<b>8</b>	<b>Interpreter: Conventional NLU for Intent Classification and Entity Recognition</b>	<b>31</b>
8.1	Cisco IoT Operations Dashboard (IoT OD) . . . . .	32
8.1.1	Edge Device Manager (EDM) . . . . .	32
8.2	Intent and Entity Definition . . . . .	33
8.3	NLU Implementation Strategies . . . . .	34
8.3.1	No-Code/Low-Code Cloud NLP Solutions . . . . .	34
8.3.2	Machine Learning (ML) Toolkits . . . . .	34
8.3.3	Conversational-AI Platforms . . . . .	35
8.3.4	Cisco MindMeld . . . . .	35
8.4	Domain, Intent, and Entity Hierarchy . . . . .	35
8.5	Data Collection . . . . .	37
8.5.1	Synthetic Data Generation . . . . .	37
8.5.2	Dataset Balancing and Splitting . . . . .	38
8.5.3	Data Organization and Annotation with MindMeld . . . . .	39
8.5.4	Non-Standard Language Challenges . . . . .	40
8.5.5	Limitations of Gazetteers and Pattern Matching . . . . .	40
8.6	Data Preparation . . . . .	41
8.7	Machine Learning Models . . . . .	41
8.7.1	Domain and Intent Classification . . . . .	42
8.7.2	Entity Recognition . . . . .	45
8.8	Performance Evaluation . . . . .	47
8.8.1	Evaluation Methodology . . . . .	47
8.8.2	Domain and Intent Classification Results . . . . .	48
8.8.3	ER Results . . . . .	51
8.9	Advantages and Limitations of Conventional NLU . . . . .	54
8.9.1	Advantages . . . . .	54
8.9.2	Limitations . . . . .	54
8.10	Chapter Summary and Practical Considerations . . . . .	55
<b>9</b>	<b>Interpreter: LLM-Based NLU for Intent Classification and Entity Recognition</b>	<b>57</b>
9.1	Suitability of LLMs for NLU Tasks . . . . .	58
9.2	Motivation for Adopting LLMs . . . . .	58

9.2.1	Shift in Project Scope . . . . .	59
9.2.2	Technical Exploration . . . . .	59
9.3	Prompt-Engineering Principles for IC and ER . . . . .	59
9.3.1	Role of Prompts . . . . .	59
9.3.2	Prompt Construction . . . . .	60
9.4	Performance Evaluation . . . . .	62
9.4.1	Performance on CCV Use-Cases . . . . .	63
9.4.2	Application to IoT Operations Dashboard Edge Device Manager Use Cases . . . . .	63
9.4.3	Noise Impact Comparison with RoBERTa . . . . .	65
9.5	Advantages and Limitations of LLMs . . . . .	65
9.5.1	Advantages . . . . .	65
9.5.2	Limitations . . . . .	66
9.6	Directions for Improvement . . . . .	67
9.6.1	Few-Shot Prompting and Fine-Tuning . . . . .	67
9.6.2	Hierarchical Separation of NLU Tasks . . . . .	68
9.6.3	Input Sanitization . . . . .	69
9.7	Chapter Summary and Practical Considerations . . . . .	69
<b>10</b>	<b>Handler</b>	<b>71</b>
10.1	Component Structure . . . . .	71
10.2	Handler Function Structure . . . . .	72
10.2.1	Template-Based Approach . . . . .	73
10.2.2	Rule-Based Approach . . . . .	73
10.2.3	LLM-Based Approach . . . . .	73
10.3	Robustness and Extensibility Enhancements . . . . .	74
10.3.1	Time-Aware Entity Handling . . . . .	74
10.3.2	Persistent Event Subscriptions . . . . .	74
10.3.3	Fuzzy Matching for Entities . . . . .	75
10.3.4	Country-Aware Entity Handling . . . . .	75
<b>11</b>	<b>Assistant</b>	<b>77</b>
11.1	Component Design and Integration . . . . .	78
11.2	Functional Extensions . . . . .	78
11.2.1	Intermediate Database for Response Caching . . . . .	79
11.2.2	LLM Temporal Limitations Addressed through Web Search . . . . .	79
11.3	Directions for Improvement Using Retrieval-Augmented Generation (RAG)	80
<b>12</b>	<b>Final Use Case Demonstrations: Webex User Viewpoint</b>	<b>82</b>
12.1	Basic Use Cases . . . . .	82
12.2	Use Cases Related to Presets . . . . .	83
12.3	Use Cases Related to Tags . . . . .	84
12.4	Use Cases Related to Baselines . . . . .	85
12.5	Use Cases Related to Groups . . . . .	86
12.6	Use Cases Related to Sensors . . . . .	86
12.7	Use Cases Related to Devices . . . . .	87
12.8	Use Cases Related to Activities . . . . .	87
12.9	Use Cases Related to Vulnerabilities . . . . .	87
12.10	Use Cases Related to Events . . . . .	90

12.11	Use Cases Related to Disconnection . . . . .	90
<b>13</b>	<b>Conclusion and Reflection</b>	<b>92</b>
13.1	System Achievements and Design Insights . . . . .	92
13.2	Limitations and Future Work . . . . .	93
	<b>Bibliography</b>	<b>95</b>
	<b>Appendices</b>	<b>103</b>
<b>A</b>	<b>Webex Bot Creation</b>	<b>103</b>
<b>B</b>	<b>Webex Bot Webhook Configuration</b>	<b>105</b>
<b>C</b>	<b>Performance Evaluation Tables</b>	<b>107</b>
C.1	DC - Conventional NLU Interpreter . . . . .	107
C.1.1	Model Comparison Summary . . . . .	107
C.1.2	RoBERTa . . . . .	107
C.1.3	Logistic Regression (LR) . . . . .	108
C.1.4	Support Vector Machine (SVM) . . . . .	108
C.1.5	Random Forest (RF) . . . . .	109
C.1.6	Decision Tree (DT) . . . . .	109
C.2	IC - Conventional NLU Interpreter . . . . .	110
C.2.1	configuration-Domained Intents . . . . .	110
C.2.2	device-Domained Intents . . . . .	111
C.2.3	greeting-Domained Intents . . . . .	113
C.2.4	operation-Domained Intents . . . . .	116
C.3	ER - Conventional NLU Interpreter . . . . .	118
C.3.1	Per-Configuration Average F1-Scores for Entity Recognition Models	118
C.3.2	Train: Random Device Names, Test: Random Device Names . . . .	118
C.3.3	Train: Random Device Names, Test: Realistic Device Names . . . .	118
C.3.4	Train: Realistic Device Names, Test: Random Device Names . . . .	119
C.3.5	Train: Realistic Device Names, Test: Realistic Device Names . . . .	119
C.3.6	Train: Mixed Device Names, Test: Random Device Names . . . . .	119
C.3.7	Train: Mixed Device Names, Test: Realistic Device Names . . . . .	120
C.4	IC - RoBERTa vs. GPT-4.1 . . . . .	120
C.4.1	configuration-Domained Intents . . . . .	120
C.4.2	device-Domained Intents . . . . .	120
C.4.3	greeting-Domained Intents . . . . .	120
C.4.4	operation-Domained Intents . . . . .	121
C.5	IC & ER - CCV Use Cases (LLM-Based NLU Interpreter) . . . . .	121
C.6	Robustness - greet Intent (Noisy Test) . . . . .	121

# Acronyms

- AI** Artificial Intelligence. 1–3, 14, 16, 26, 35, 56, 59, 92, 94
- API** Application Programming Interface. 14, 15, 23–29, 33, 34, 67, 69, 73, 80, 93
- BERT** Bidirectional Encoder Representations from Transformers. 40, 41, 44, 45, 47, 53–55, 58, 59, 66–69
- BiLSTM** Bidirectional LSTM. 46
- BPE** Byte Pair Encoding. 45, 58
- CCV** Cisco Cyber Vision. 2–14, 22, 26–29, 31, 33, 54, 56, 57, 59, 65, 66, 73–75, 79, 80, 83, 87, 92–94
- CLM** Causal Language Modeling. 58
- CNN** Convolutional Neural Networks. 34, 41, 46
- CNN-LSTM** Character-Level CNN Followed by Word-Level LSTM. 45, 47, 53, 54, 92
- CRF** Conditional Random Field. 41, 47
- CVE** Common Vulnerabilities and Exposures. 12, 75, 79, 80
- CVSS** Common Vulnerability Scoring System. 12, 19, 87
- DC** Domain Classification. 35, 36, 38, 41, 42, 45, 47–49, 51, 52, 54–56, 60, 68, 93
- DT** Decision Tree. 41–44, 46, 48, 50, 58
- ER** Entity Recognition. 17, 18, 22, 27, 31, 34–36, 39–42, 45–47, 51–58, 60, 63–66, 68, 69, 74, 92, 93
- GenAI** Generative AI. 4, 57, 59, 67, 69, 73, 77, 79, 93
- GloVe** Global Vectors for Word Representation. 46
- GPT** Generative Pre-Trained Transformer. 57–60, 64–68, 70, 73, 78, 92
- HMI** Human-Machine Interface. 6, 7
- IC** Intent Classification. 17, 18, 22, 27, 31, 33–36, 41, 42, 44, 45, 47–49, 51, 52, 54–58, 60, 63–65, 68, 69, 92, 93

**ICS** Industrial Control System. 5–9

**IOBES** Inside-Outside-Begin-End-Single. 45, 46

**IoT** Internet of Things. 5, 32, 59

**IoT OD EDM** Cisco IoT Operations Dashboard Edge Device Manager. 4, 31–33, 52, 54, 56, 57, 59, 63–65, 68

**IT** Information Technology. 1, 5–7

**JSON** JavaScript Object Notation. 24, 25, 28, 60, 62, 63, 66, 67, 69

**LLM** Large Language Model. 4, 17, 22, 31, 38, 40, 56–60, 63–69, 72–74, 77–82, 92–94

**LR** Logistic Regression. 34, 41–44, 46, 48–51, 54, 55, 58, 92

**LSTM** Long Short-Term Memory Networks. 34, 41, 45–47, 53

**MEMM** Maximum Entropy Markov Model. 41, 45, 46, 53, 55, 58

**ML** Machine Learning. 4, 16, 22, 31, 32, 34, 35, 37, 40–43, 54, 55, 57, 59, 64

**MLM** Masked Language Modeling. 44

**NLG** Natural Language Generation. 16–18, 59, 72, 73, 77

**NLP** Natural Language Processing. 16, 18, 34, 35, 57, 58, 92

**NLU** Natural Language Understanding. 16–18, 27, 31–35, 40–42, 45, 54–60, 68, 69, 92

**NSP** Next Sentence Prediction. 44, 45

**OOV** Out-of-Vocabulary. 43, 46

**OT** Operational Technology. 2, 3, 5–9, 13, 59, 92

**PLC** Programmable Logic Controller. 6, 7, 9

**RAG** Retrieval-Augmented Generation. 78, 80, 81, 93

**RC** Role Classification. 36, 54

**RF** Random Forest. 41–44, 48–50

**RNN** Recurrent Neural Network. 46

**RP** Reverse Proxy. 23

**SCADA** Supervisory Control and Data Acquisition. 6, 9

**SVM** Support Vector Machine. 34, 41–43, 48, 50, 51

**UI** User Interface. 3, 8–14, 26, 34, 74, 83, 93, 94

# Chapter 1

## Introduction

### 1.1 Origins of Chatbots: Early Milestones in Tech and Culture

“October 13, 1966, before there were personal computers, NBC aired episode six of the very first season of *Star Trek* that day. For the first time, a Starfleet officer, Captain James T. Kirk, turns to a console on the USS Enterprise and says, ‘Computer, start.’ The computer listens. As the humans in the room speak to it, the computer speaks back, answering questions, responding to statements, and delivering information” [1, p. 9].

That same year, MIT professor Joseph Weizenbaum created the first chatbot known as ELIZA, which was only able to recognize certain keywords to respond accordingly [2].

Those two events marked an early milestone in the development of chatbots, which have emerged as a pivotal technology within today’s digital landscape, responding to the imperative of simplifying complex tasks and enhancing everyday convenience [1, pp. 9–10].

While there is no strict classification, chatbots can generally be divided into two main families: rule-based chatbots, which can be considered descendants of ELIZA and are designed to respond to a predefined set of inputs, and chatbots powered with Artificial Intelligence (AI), reminiscent of the vision portrayed by the speaking computer in *Star Trek* in 1966—a dream of “communicating with our technology the same way we communicate with each other—through conversation” [1, p. 9]. The latter family aims to achieve this dream by employing AI techniques to enable more dynamic, context-aware interactions and, more generally, mimic human language [3, 4], allowing humans to catch up with the futuristic visions portrayed in *Star Trek*.

### 1.2 Objectives of the Thesis

Accessibility to technology has been a paramount concern for numerous stakeholders in the Information Technology (IT) industry, as it provides businesses with access to larger markets [5]. As the world becomes increasingly interconnected—illustrated in Figure 1.1, which shows that 68% of the global population was connected in 2024, according to



the International Telecommunication Union (ITU) [6]—the demographics engaging with technology are growing progressively more diverse [7].

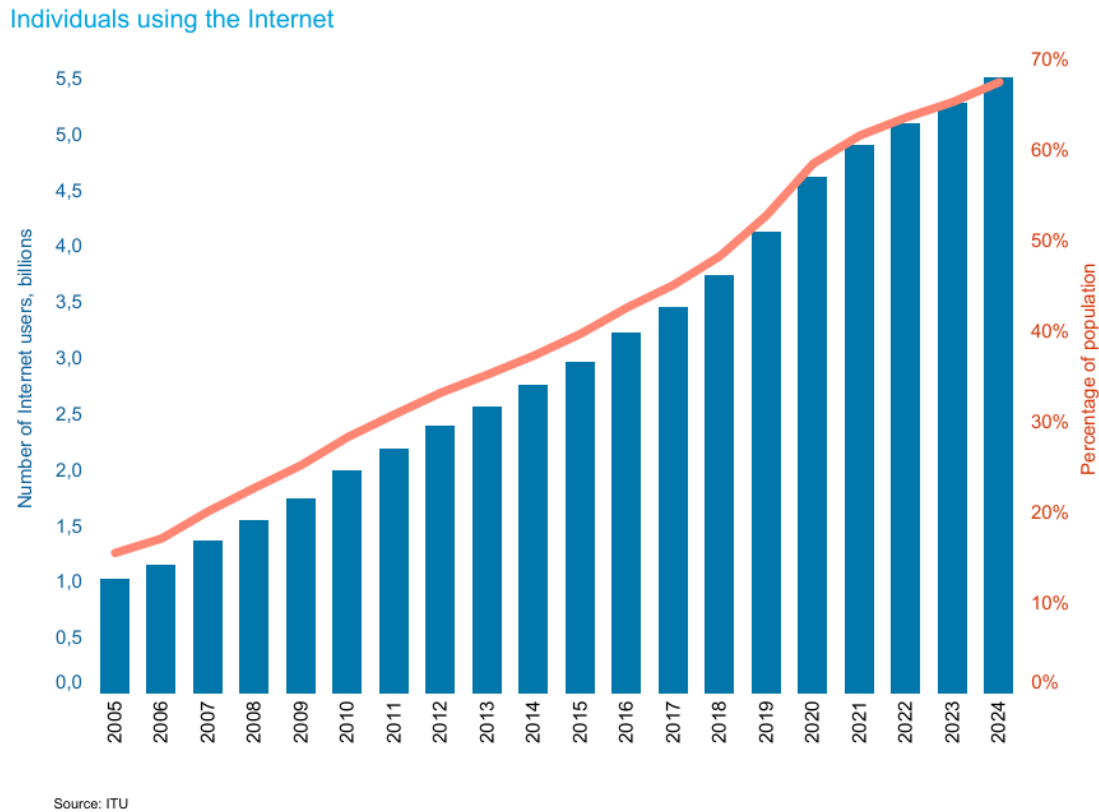


Figure 1.1. Number of individuals using the Internet from 2005 to 2024 [6].

To ensure technology accessibility, one must first understand what accessibility means. According to the Cambridge Dictionary, being accessible is the quality of being usable, understandable, and enjoyable by everyone [8]. The World Wide Web Consortium (W3C) also emphasizes the importance of perceivability, which ensures that information is presented in a way that users can perceive through their senses [9].

In light of these principles, AI-powered chatbots have emerged as a prominent solution to numerous accessibility challenges. They are engineered to be easy to use, can be understood by a wide range of individuals as they tend to be programmed to use plain and understandable language, avoiding complex jargon or convoluted sentences, and interact with users through visual text or auditory interfaces, thereby aligning with the perceivability principle.

It is within this context of enhancing accessibility to technology that this project lies itself. Specifically, it centres around the development of a chatbot—built for Webex, a collaboration platform—that bridges the gap between the user and Cisco Cyber Vision (CCV), a security platform specifically designed for Operational Technology (OT) environments. Users interact with the chatbot via a Webex space or direct messaging. A Webex bot serves as the interface, forwarding messages to the chatbot server that implements the conversational capabilities. The interaction is designed to be user-friendly, employing

natural human language and enabling users to ask questions such as: “What devices are in my network?”, “What activities happened between devices  $X$  and  $Y$  this week?”, “What vulnerabilities does device  $X$  have?”, “Explain vulnerability  $X$ ”, and so on.

Instead of relying on the existing web User Interface (UI) to interact with CCV, the chatbot offers a more accessible approach, allowing users to engage with it using natural language within a familiar, friendly, and intuitive text-based environment. By harnessing the benefits offered by conversational AI, such as simplifying complex jargon, CCV broadens its reach to a wider audience, including those with less expertise. This approach offers a more intuitive and accessible way to manage security in industrial OT environments.

Throughout this thesis, the term *bot* is used to refer specifically to the Webex virtual user—an entity registered on the Webex platform that interacts with users via spaces or direct messaging, and routes user messages through webhooks, as will be detailed in Section 2.4. In contrast, the term *chatbot* is used to refer to the backend component developed as part of this thesis, which is responsible for implementing the system’s conversational intelligence. This distinction highlights the separation of responsibilities between the Webex platform interface (bot) and the conversational logic component (chatbot).

## 1.3 Structure of the Thesis

This thesis has two primary objectives: to describe the key design aspects of the implemented solution and to explain the rationale behind the decisions made during its development.

Chapter 2 lays the groundwork by providing an overview of Cisco Systems, the CCV platform, the OT environment in which it operates, and Webex. This context helps situate the chatbot within its broader operational ecosystem.

Next, Chapter 3 investigates the chatbot’s conversational requirements, outlining the core principles of conversational AI that enable it to understand and respond appropriately to human input.

With a clear understanding of both the integration environment and conversational requirements, Chapter 4 defines the chatbot’s specific use cases. This chapter delineates the functional scope of the system and establishes the boundaries that guide its design.

The remainder of the thesis adopts a component-oriented structure to document the design and implementation phase. Chapters are organized to mirror the end-to-end flow of a Webex user message—starting from initial user input to the generation and delivery of a response. Following a high-level overview of the application’s architecture in Chapter 5, each subsequent chapter focuses on a distinct component, examining the design choices and implementation details in sequence.

The process begins with the setup of the *Chatbot Server* in Chapter 6, which is responsible for capturing incoming messages, coordinating the various components to execute the necessary logic, and ultimately delivering a response that addresses the specific use case indicated by the user’s input.

To provide information from the CCV platform, the chatbot must communicate with it on the user’s behalf, which necessitates authentication. This process is handled securely and reliably by the *Authenticator* component, as detailed in Chapter 7.

Once authenticated, the user’s message is passed to the *Interpreter* component, responsible for analyzing the message and producing a structured representation of its meaning. This component is implemented using two distinct approaches, described in Chapters 8 and 9. The first solution integrates with Cisco IoT Operations Dashboard Edge Device Manager (IoT OD EDM) and relies on a conventional<sup>1</sup> Machine Learning (ML) approach. A subsequent implementation leverages Large Language Models (LLMs)—a subset of Generative AI (GenAI) technologies—and targets CCV<sup>2</sup>. Based on the insights gained from the first Interpreter implementation, we analyze its advantages and limitations. The second, LLM-based implementation addresses these shortcomings while leveraging the advanced reasoning capabilities of LLMs.

The output from the Interpreter is then forwarded to the *Handler*, described in Chapter 10, which determines the appropriate action in response to the user’s message and implements logic to enhance the overall user experience by supporting greater flexibility in natural language input.

The final component is the *Assistant*, presented in Chapter 11. This module leverages an LLM and prompt engineering techniques to generate detailed responses tailored to the supported use cases.

The thesis concludes with a demonstration of the chatbot’s capabilities from the user’s perspective in Chapter 12, followed by Chapter 13, which provides a comprehensive reflection on the system’s achievements, design decisions, and implementation insights. It also discusses current limitations, outlines potential extensions, and situates the work within the broader context of modern chatbot development and future research directions.

The source code of the developed chatbot is publicly available on GitHub [10].

---

<sup>1</sup>In this thesis, “conventional” refers to widely adopted, pre-LLM approaches that do not rely on large-scale generative models. These methods are typically task-specific and require explicit model training and feature design.

<sup>2</sup>The initial integration with IoT OD EDM was based on its original role as the chatbot’s target application. However, following Cisco’s decision to discontinue IoT OD EDM, the project scope shifted to CCV. Although CCV serves a different purpose, foundational components developed for IoT OD EDM provided valuable insights and informed the second implementation. This shift also facilitated an assessment of the initial approach’s limitations and the development of a more robust solution.

# Chapter 2

## Integration Environment

This chapter outlines the key stakeholders and components central to this work. Section 2.1 introduces Cisco Systems, the organization for which the chatbot is being developed. Sections 2.2 and 2.3 examine the CCV platform in the context of OT environments. Section 2.4 discusses Cisco Webex and its bot integration capabilities, which serve as the platform supporting the chatbot’s functionality.

### 2.1 Cisco Systems

As one of the most prominent IT companies in the world, Cisco Systems (hereafter referred to as *Cisco*) aims to provide technology products and services that respond to the challenges of the modern digital era. Founded in 1984, Cisco has evolved from its initial focus on networking into a multifaceted technology powerhouse that shapes the digital landscape [11]. Today, Cisco stands as a key actor in various areas, including Security, Cloud and Data Centers, Internet of Things (IoT), and Collaboration [12].

### 2.2 Operational Technology (OT) and Industrial Control Systems (ICS)

As opposed to IT, which refers to the use of computing devices—including computers, storage devices, networking hardware, software applications, and telecommunication equipment—to create, process, store, secure, and distribute digital information [13, 14], OT focuses on the hardware and software that monitor and control industrial equipment such as machinery, sensors, actuators, and other devices used in industrial settings [15, 16]. In other words, IT systems manage data and applications, while OT devices control the physical world.

At the heart of OT are Industrial Control Systems (ICSs), a broad term encompassing various devices used to monitor and control industrial equipment [17]. This section explores each level of the Purdue Enterprise Reference Architecture (PERA), which we will refer to as the *Purdue Model*—a framework that organizes an ICS network into distinct levels, each with specific roles, technologies, and security considerations [18]. Although the Purdue Model is often criticized as outdated in today’s OT landscape—where the boundaries between levels have blurred or even disappeared [19]—it remains relevant in this thesis

for two key reasons: it simplifies the understanding of the ICS ecosystem by abstracting away overly complex details irrelevant to this work, and its principles have influenced the design of CCV by offering guidance on where to apply security measures effectively [20]. By understanding the purpose of the Purdue Model’s levels, we can better identify where CCV fits within the OT landscape.

### **2.2.1 Level 0: Physical Process**

The physical process level is the layer where industrial processes are carried out by equipment like sensors and actuators. Sensors measure physical parameters such as temperature, humidity, and more, while actuators perform physical actions based on control signals, like manipulating valves, motors, and similar devices. These components work together in a complementary manner: sensors provide feedback to controllers, which process the data and issue commands to actuators in order to regulate and optimize the process. Controllers constitute Level 1 of the Purdue Model [18].

### **2.2.2 Level 1: Basic Control**

The basic control level is responsible for executing control logic and managing input and output for Level 0 devices. While not strictly limited to it, the most common device is the Programmable Logic Controller (PLC), which automatically collects data from sensors and instruct actuators in the level below to execute the appropriate commands [18].

### **2.2.3 Level 2: Supervisory Control**

The supervisory control level is where humans primarily interact with the OT environment using equipment such as Human-Machine Interfaces (HMIs) and Supervisory Control and Data Acquisition (SCADA) systems. HMIs connect to lower-level equipment, enabling both monitoring and control. SCADA—a broader system that encompasses HMIs—collects data from potentially all devices in the lower levels of the ICS network to provide comprehensive oversight and forwards relevant information to upper layers, as defined by the Purdue Model [18, 21].

### **2.2.4 Level 3: Manufacturing Operations**

The manufacturing operations level—the highest layer within the OT side of the Purdue Model—is responsible for collecting and storing data from various sources within the ICS network in specialized databases known as Historians, enabling subsequent data analysis. Another essential component at this level is the Manufacturing Execution System (MES), which interprets data received from lower layers—particularly from SCADA systems—to support informed control and production decisions [18, 22, 23].

### **2.2.5 Beyond Level 3**

Level 3 marks the final layer of the OT environment. Beyond it lies Level 4, which is separated by a Demilitarized Zone (DMZ) for security, typically enforced through firewalls. Level 4 belongs to the IT domain and is responsible for orchestrating lower layers, hosting business applications, and optionally enabling internet connectivity. Some versions of the Purdue model also define a Level 5, which represents the enterprise network, encompassing

corporate IT systems such as Enterprise Resource Planning (ERP) platforms and cloud services [19].

## 2.3 Cisco Cyber Vision (CCV)

CCV is a security platform designed for OT environments. It provides insights into all assets deployed within a network, monitors the communications occurring between those assets, and identifies vulnerabilities [24]. CCV positions itself inside the ICS network outlined in the previous section by employing a two-tier architecture consisting of a CCV center and sensors or a three-tier architecture where a global CCV center collects information from multiple centers each having limited reach to their respective sensors [25]. The center serves as the primary application for user interaction, while the sensors perform deep packet inspection and collect information from the OT environment to transmit to the center. For simplicity, a two-tier architecture with a single center is used, as it sufficiently covers the scope of this thesis.

### 2.3.1 CCV Sensors

CCV sensors are deployed at the edges of the OT network, close to HMIs, PLCs, and other industrial devices corresponding to Levels 0, 1, 2, and 3 of the Purdue Model [25], as shown in Figure 2.1. These sensors can be deployed either in-line, where they are inserted between two network segments and become part of the traffic path, or connected to a network switch configured to mirror traffic using a Switched Port Analyzer (SPAN), an efficient monitoring setup. The latter is preferred, as it leaves the network topology unaltered and keeps the sensors in a purely passive role.

Captured traffic is continuously analyzed and decoded by Cisco’s Deep Packet Inspection (DPI) engine within each sensor. The extracted metadata, including relevant insights from the analyzed traffic, is then transmitted to the CCV center [25].

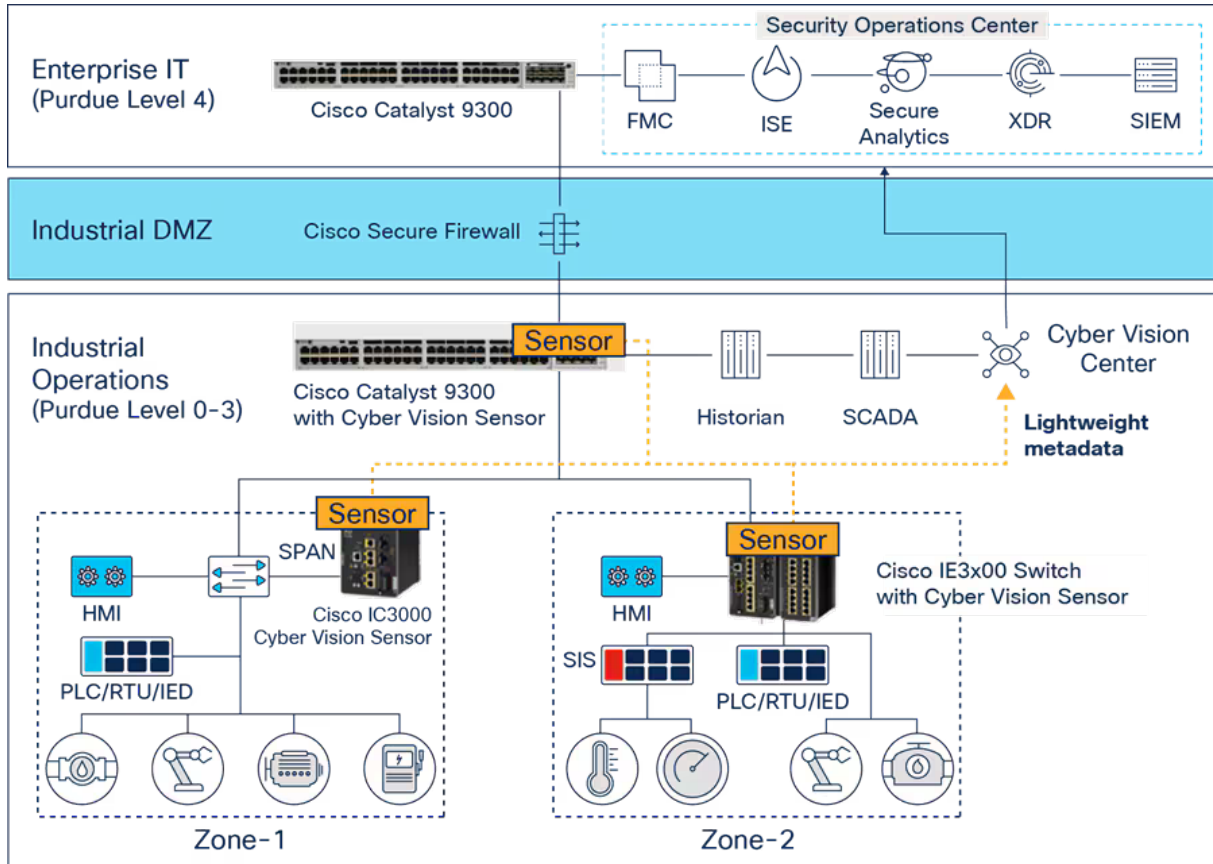


Figure 2.1. High-level architecture of CCV [26]. Sensors deployed at various OT levels monitor traffic between industrial components and extract metadata. This metadata is transmitted to the CCV center at Purdue Level 3, where it is analyzed. The CCV center integrates with security applications at the enterprise level (Purdue Level 4) to provide comprehensive security insights to the user.

### 2.3.2 CCV Center

The CCV center is designed with two primary objectives: visibility and security. It provides visibility into the network by enabling asset tracking and monitoring of communications between devices. Furthermore, it enhances security by identifying vulnerabilities and assigning risk scores to help prioritize mitigation efforts [24].

As the specific configuration details for integrating CCV into an ICS network are beyond the scope of this work, we assume an environment where the system is already deployed, with its sensors sending data to the center. The center is accessible through a web UI at a specified IP address, with authentication handled via email and password.

To fully understand the benefits of CCV, the following section provides an overview of its platform architecture and key functionalities. This understanding will inform the development of the chatbot’s functional scope and overall solution.

### 2.3.3 Concepts and Functions of the CCV Center

The CCV center presents a collection of concepts that, together, enhance network visibility and yield meaningful security insights. This section examines several of these key concepts—



relevant to the chatbot’s functional scope—and their representation within the CCV center’s web UI.

## Device

A device is a physical machine present in the OT environment. It can be any component from the ICS, including industrial sensors or actuators, PLCs, SCADA stations, historians, and switches—but also general-purpose systems such as servers and personal computers (PCs), which, while not inherently industrial, are commonly part of such environments [27].

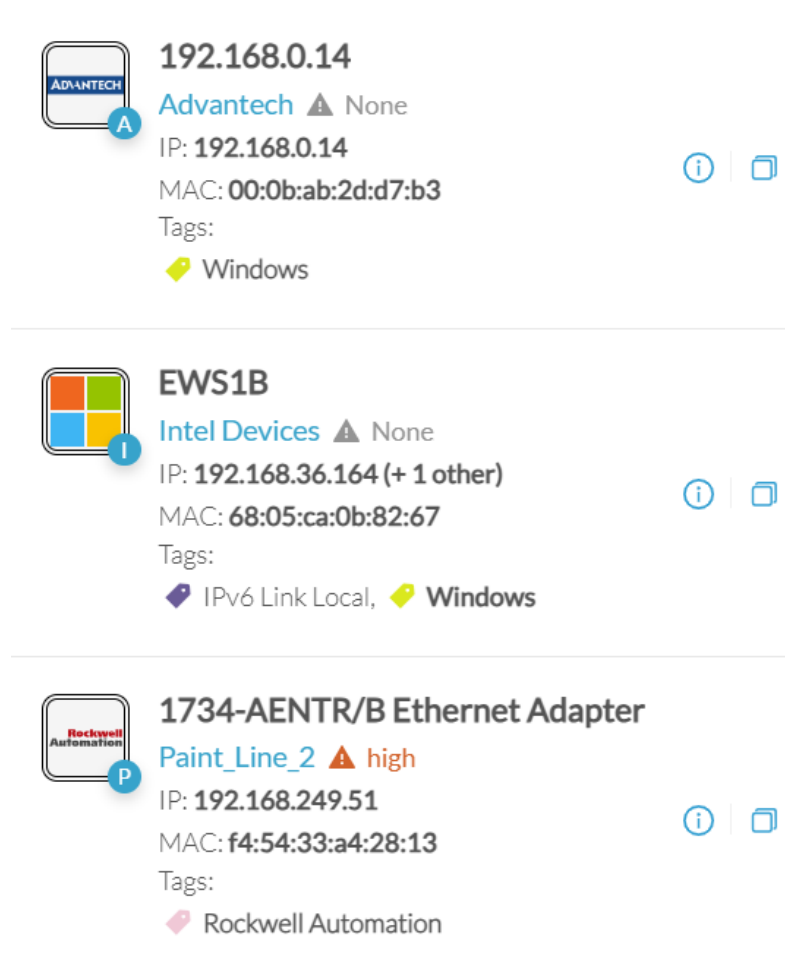


Figure 2.2. Overview of devices as displayed in the CCV center’s web UI, including key attributes such as IP address, MAC address, and tags.

## Component

Each device is composed of components. Components are parts of devices, representing a specific functional or network-facing element. Examples of components are the network interface of a PLC, an IP address associated with a SCADA, a broadcast or multicast address. In other words, components are the building blocks of a device, each contributing to its specific tasks [28].



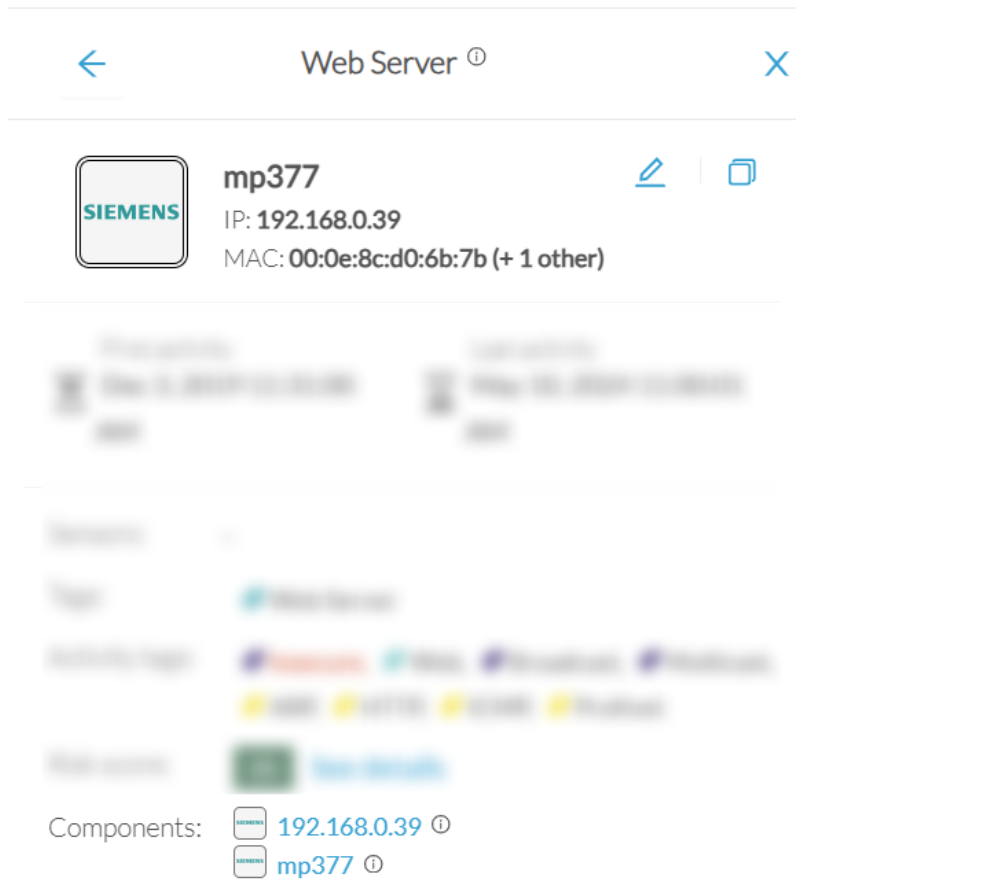


Figure 2.3. Overview of device components as displayed in the CCV center’s web UI.

## Group

A group is a user-defined method for organizing devices and components based on criteria such as location, type, or other relevant factors [29].

## Flow

A flow is a single communication between two components. It has a direction, involves the relevant ports of each component, includes start and end timestamps, and records both the number of packets exchanged and the total bytes transferred [30].

## Activity

While visibility into individual flows offers advantages, storing all flows can quickly lead to substantial storage demands. As a result, users primarily interact with activities—aggregated communications between two devices that share a common purpose—each consolidating multiple flows into a single entity. Activities retain the same metadata as individual flows and maintain a list of the flows they aggregate. However, for resource efficiency, this list is not necessarily stored persistently [31].


Device	Device	Flows	Packets	Volume
DESKTOP-GBJUF2N	1734-AENTR/B Ethernet Adapter	~70	47882	5.24 MB
DESKTOP-GBJUF2N	192.168.249.255	~10	39	5.41 kB

Figure 2.4. Overview of activities as displayed in the CCV center’s web UI, including aggregated flow data such as number of flows, packets, and total volume exchanged between devices.

## Tag

Tags, generated at the flow level or the component level, are metadata that describe their function. These tags are synthesized at the activity level and the device level, respectively. These tags are predefined in CCV and are automatically derived from the analysis of metadata and behavior patterns observed in the network, aiding in the classification and organization of data for easier interpretation. For instance, a flow can be tagged based on the source and destination addresses, the protocol in use, and the type of data being exchanged. Similarly, a component can be tagged according to its type and purpose [32].

←
Web Server ⓘ
×



**mp377**
✎ 📄

IP: 192.168.0.39  
MAC: 00:0e:8c:d0:6b:7b (+ 1 other)

Tags:

Activity tags:

Web Server

Insecure, Web, Broadcast, Multicast, ARP, HTTP, ICMP, Profinet

Figure 2.5. Overview of device and activity tags as displayed in the CCV center’s web UI, including classifications such as role, protocol usage, and security relevance.

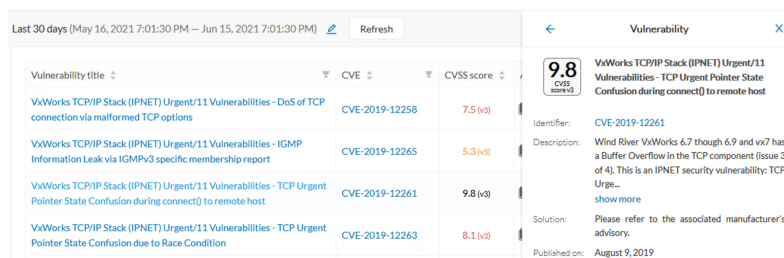
## Vulnerability

A vulnerability is a weakness detected in a component or device that could be exploited by an attacker. The CCV center leverages a vulnerability knowledge base—a centralized repository of detection rules derived from trusted sources. A rule is triggered when a property of a component or device, such as its firmware version, protocol usage, or other attributes, matches the criteria defined in the rule. When a match occurs, the corresponding vulnerability is flagged and made visible to users, enabling proactive risk mitigation [33].

A vulnerability is primarily defined by four attributes:

1. **Descriptive information** about the vulnerability.
2. A **recommended solution** to mitigate or remediate the issue.
3. A **severity score**, known as the Common Vulnerability Scoring System (CVSS) score, which ranges from 0 to 10. Scores of 7 or higher typically indicate high or critical vulnerabilities [34].
4. A **unique identifier**, known as a Common Vulnerabilities and Exposures (CVE), which begins with “*CVE*”, followed by the year of public disclosure and a unique ID for that year—for example, *CVE-2019-12261* [33].

Based on device tags, the likelihood of compromise, exposure to the internet, and detected vulnerabilities—along with their CVSS scores—a risk score is assigned to each device. This score provides an indication of the device’s overall security posture and operational health [35].



The screenshot shows a web interface for viewing vulnerabilities. On the left, a table lists four vulnerabilities with columns for title, CVE ID, and CVSS score. On the right, a detailed view for CVE-2019-12261 is shown, including its identifier, description, solution, and publication date.

Vulnerability title	CVE	CVSS score
VxWorks TCP/IP Stack (IPNET) Urgent/11 Vulnerabilities - DoS of TCP connection via malformed TCP options	CVE-2019-12258	7.5 (v)d
VxWorks TCP/IP Stack (IPNET) Urgent/11 Vulnerabilities - IGMP Information Leak via IGMPv3 specific membership report	CVE-2019-12265	5.3 (v)d
VxWorks TCP/IP Stack (IPNET) Urgent/11 Vulnerabilities - TCP Urgent Pointer State Confusion during connect() to remote host	CVE-2019-12261	9.8 (v)d
VxWorks TCP/IP Stack (IPNET) Urgent/11 Vulnerabilities - TCP Urgent Pointer State Confusion due to Race Condition	CVE-2019-12263	8.1 (v)d

**Vulnerability**

**9.8** CVSS score (v3)

**Identifier:** CVE-2019-12261

**Description:** Wind River VxWorks 6.7 through 6.9 and vx7 has a Buffer Overflow in the TCP component (issue 3 of 4). This is an IPNET security vulnerability: TCP Urgent...

**Solution:** Please refer to the associated manufacturer's advisory.

**Published on:** August 9, 2019

Figure 2.6. Overview of identified vulnerabilities in the CCV center’s web UI, including CVE identifiers, CVSS scores, and short descriptions of each issue.

## Preset

The preset is a curated data filter designed to enhance network visibility by focusing on specific metadata. It acts as a contextual lens, enabling users to efficiently sift through large volumes of network data and isolate meaningful insights [36].

A preset filters data based on several criteria: the network subnet to monitor; the devices included, determined by tags, risk scores, or group membership; the activities to track, based on their tags; and the sensors providing the associated metadata [37].

Users can either apply the default presets provided by CCV or define custom ones. The majority of use cases in the CCV center are carried out through presets.

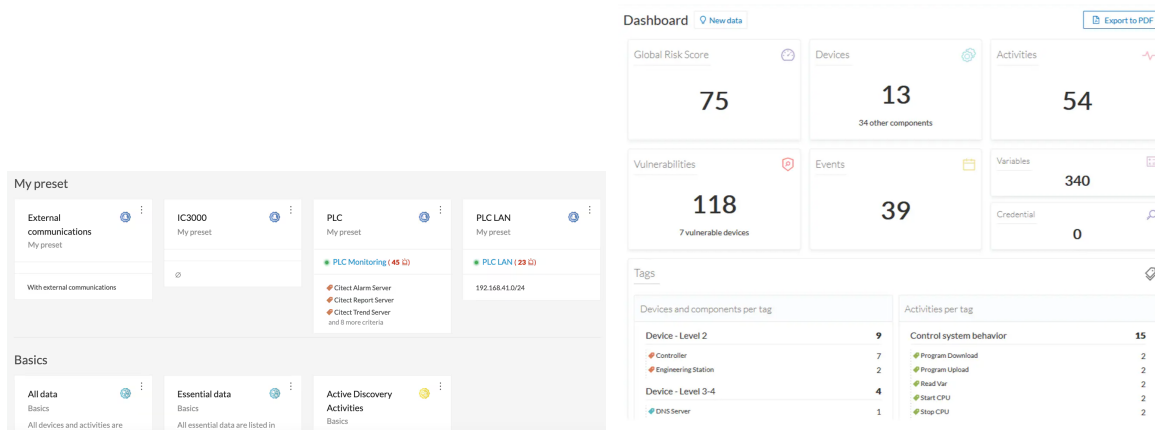


Figure 2.7. Overview of presets in the CCV center’s web UI. The left panel displays a list of available presets, while the right panel shows the dashboard view for a selected preset, summarizing key metrics such as risk score, devices, vulnerabilities, and tags.

## Preset Baseline

A preset baseline is a snapshot of the network within a specific preset, representing what is considered normal for that segment of the network. More specifically, a baseline captures the typical state of the network during a given period or under certain operating conditions, serving as a reference point. Any previously unseen protocols, newly connected devices, or other deviations will be identified as baseline differences and brought to the user’s attention. These differences can then be acknowledged as legitimate parts of the network or reported as potential issues [38].

### 2.3.4 Limitations of the CCV Center Web User Interface

While CCV is extensively used in OT environments, it presents several limitations that the chatbot aims to mitigate.

#### Limitation 1: Learning Curve and Information Accessibility

Effectively using the CCV center requires a high level of technical expertise. Navigating through smart filters, interpreting tags, and addressing vulnerabilities demands skills that many users may not possess. This expertise gap can create a significant barrier to entry and introduces a steep learning curve, particularly for new or non-specialist users.

#### Limitation 2: Limited Guidance

Although the CCV center provides explanations for tags, vulnerabilities, and associated solutions, these are often insufficient or superficial. As a result, users frequently need to conduct additional research outside the platform. This workflow is inefficient and detracts from the overall user experience.

#### Limitation 3: Fragmented Team Experience

While CCV centers are shared across individuals, the user experience remains largely personal. Information sharing and collaboration often occur outside the platform, through

external tools such as Webex. This fragmentation limits the center’s effectiveness as a central knowledge and coordination hub.

### 2.3.5 CCV Center Application Programming Interface (API)

Using an Application Programming Interface (API) token and the IP address of the CCV center, all actions available through the web UI can also be performed via the API, which exposes a RESTful interface for interacting with the system programmatically [39]. While a detailed exploration of the API’s capabilities is not necessary, these endpoints will be extensively used by the chatbot to retrieve relevant network information and respond to user queries.

## 2.4 Cisco Webex

Meet from anywhere and host engaging interactive events with audiences of any size while ensuring security and privacy, high-quality video and audio, and seamless integration with other productivity tools and platforms. This is the bundle Cisco has been offering since 2007 with its highly-used collaboration platform: Webex. Whether for personal or professional purposes, Webex remains a trusted choice for virtual collaboration [40].

Among the many features offered by Webex, one notable capability is bot integration. These bots can operate within spaces—digital workspaces designed for collaborative communication among multiple users—or through direct messaging, enabling seamless service integration within the Webex platform [41].

### 2.4.1 Bot Integration in Webex

Bots have become an integral part of the Webex ecosystem, offering functionalities ranging from notifications for specific events like GitHub commits [42] to automating routine tasks with pre-defined reminders [43]. They can also function as text-based remote controls for external services such as Jira [44] and LucidChart [45], leveraging conversational AI to interact in a user-friendly manner. This project focuses on developing such an assistant-type chatbot.

Bots can be interacted with in a manner akin to human users: they can be messaged directly or added to a space, where multiple users and bots can communicate. The main difference is that, in the latter case, a bot only has access to messages in which it is mentioned and interacts with Webex through its API rather than the standard user interface. Webex provides a RESTful API, which the chatbot uses to send and receive messages programmatically.

#### Webex Bot Creation

Before going further, one needs to create a Webex bot, achieved through the Webex Developer Portal [41]. The exact process is described in Appendix A. The result is a bot defined by a unique name, a unique username (used to invite it for direct messaging or to a space), and an access token, which is required for setting up webhooks.

## Webex Webhook

Webhooks in Webex act as HTTP callbacks to a specified URL, enabling real-time data transmission [46]. When a message is sent to a bot, Webex forwards this message to a server at a predetermined URL, which handles the chatbot's logic. Webhooks are configured using the Webex Webhooks API [47] using the bot's access token and are identified by a unique identifier upon creation. Each webhook includes a target URL, where messages are sent, and a resource type that specifies the kind of resources forwarded to the webhook. Two resource types are particularly important for the chatbot developed in this work:

1. **Message** resource type: Represents plain text messages sent in Webex chats, serving as the primary method of communication with the bot.
2. **Attachment Action** resource type: Triggered when a user interacts with a Webex card.

The configuration and specific use cases of these two webhook types are detailed in Chapters 6 and 7, which cover the setup of the Chatbot Server and Authenticator components, respectively.

# Chapter 3

## Conversational Requirements

A key aspect of the chatbot is its ability to understand and generate natural language. It must accurately interpret the user’s intent and produce an appropriate response within a natural language framework. This framework must account for challenges such as ambiguous wording, typographical errors, and irrelevant information in user inputs. This chapter outlines the main conversational requirements the chatbot must satisfy. The concrete implementation of these requirements will be detailed in later dedicated chapters.

### 3.1 Conversational Artificial Intelligence (AI)

Conversational AI is a subset of AI focused on understanding, processing, and responding to human language. The chatbot developed in this thesis serves as an example of this technology [48]. This capability is enabled by Natural Language Processing (NLP), a field of AI that uses ML to equip software with the ability to understand and respond to human language. Within NLP, these tasks are generally divided into two key components: Natural Language Understanding (NLU), which involves interpreting the meaning and intent behind user inputs; and Natural Language Generation (NLG), which focuses on generating coherent and contextually appropriate responses [49].

### 3.2 Natural Language Understanding (NLU)

The first essential capability of the chatbot is to understand the meaning behind a user’s message. In the context of this work, *understanding* serves a dual purpose: identifying the message’s intent and extracting relevant entities. These processes enable the chatbot to interpret user inputs accurately and generate meaningful, context-aware responses.

The *intent* conveys the user’s goal, guiding the chatbot in selecting the appropriate function to execute. *Entities* provide specific information within the message that supports the function’s execution.

For example, the message “**Explain vulnerability CVE-2019-12261**” should produce the *intent-entity* pair `explain_vulnerability` and `CVE-2019-12261`. The intent indicates the required action (explaining a vulnerability), while the entity specifies the input—in this case, the name of the vulnerability.

The following sections provide a high-level overview of how the chatbot handles Intent Classification (IC) and Entity Recognition (ER)—the two core components of NLU in this work. Their concrete implementation is discussed in Chapters 8 and 9, which detail the development of the Interpreter component.

### 3.2.1 Intent Classification (IC)

IC, sometimes referred to as intent recognition, aims to identify the user’s purpose or goal in their message. It enables the chatbot to determine the appropriate action or response based on the user’s input. This process involves mapping natural language input to a predefined set of intents, each representing a specific functionality of the system.

### 3.2.2 Entity Recognition (ER)

After classifying the user’s message, the next crucial step is to identify relevant entities that the system’s response may depend on. Entities can be either *named*—such as locations, people, or organizations—or *non-named*, such as passwords or numerical identifiers. This process is known as ER.

## 3.3 Natural Language Generation (NLG)

NLG is used to generate natural language output, which, in the context of this work, refers to the text produced by the chatbot in response to messages sent by a Webex user. The generated content is informed by the outcomes of the IC and ER stages of the NLU process and can be produced using three techniques: *template-based*, *rule-based*, and *LLM-based* approaches. This work employs all three, selecting the most suitable method for each use case in order to balance simplicity, control, and flexibility.



# Chapter 4

## Functional Scope of the Chatbot

Before delving into the specific design aspects of the chatbot, it is essential to define its functional scope. The use cases outlined in this chapter are the result of ongoing discussions with several Cisco stakeholders overseeing the project and were not all defined from the outset.

Having introduced the chatbot’s conversational requirements—namely, the concepts of IC and ER within the context of NLU, as well as NLG for generating responses—we now define the chatbot’s use cases in terms of the core NLP components they require. Specifically, each use case is characterized by the intent the chatbot must classify, any optional entities it must recognize, and the expected structure of the chatbot’s response it must generate for the originating Webex user.

### 4.1 Use-Case Definition

Each chatbot use case is defined by an intent, a set of required or optional entities, and the expected response structure. These are summarized in Table 4.1.

Additionally, the chatbot must clearly communicate errors—whether due to misclassified intents, missing or invalid entities, or other processing issues.

Table 4.1. Chatbot use cases with corresponding intents, entities, and response structures.

Use Case	Intent	Entities	Response
The user greets the chatbot.	<b>greet</b>	Not applicable	A greeting message.
The user ends the conversation with the chatbot.	<b>exit</b>	Not applicable	A farewell message.
The user requests information about the chatbot’s available functionalities.	<b>help</b>	Not applicable	A description of the chatbot’s use cases.
The user requests a list of presets, optionally filtered by a specific set of tags.	<b>list_presets</b>	Optional <b>tags</b>	The list of presets, optionally filtered by the specified tags.
The user requests details for a specific preset.	<b>preset_details</b>	Required <b>preset_label</b>	Detailed information about the specified preset.
The user activates a preset.	<b>activate_preset</b>	Required <b>preset_label</b>	A confirmation message indicating preset activation.

Use Case	Intent	Entities	Response
The user deactivates the currently active preset.	<code>deactivate_preset</code>	Not applicable	A confirmation message indicating preset deactivation.
The user requests a list of available tags.	<code>list_tags</code>	Not applicable	A list of tags.
The user requests an explanation for a specific tag.	<code>explain_tag</code>	Required <code>tag_label</code>	An explanation of the requested tag.
The user requests a list of all baselines.	<code>list_baselines</code>	Not applicable	A list of baselines.
The user requests details about a specific baseline.	<code>baseline_details</code>	Required <code>baseline_label</code>	Detailed information about the requested baseline.
The user requests a list of devices, optionally filtered by vendor country of origin.	<code>list_devices_components</code>	Optional <code>vendor_country</code>	A list of devices, optionally filtered by country of origin.
The user requests details for a specific device or component.	<code>device_component_details</code>	Required <code>device_label</code>	Detailed information about the requested device or component.
The user requests a list of groups.	<code>list_groups</code>	Not applicable	A list of groups.
The user requests details for a specific group.	<code>group_details</code>	Required <code>group_label</code>	Detailed information about the requested group.
The user requests a list of sensors.	<code>list_sensors</code>	Not applicable	A list of sensors.
The user requests a list of activities, optionally filtered by start date, end date, and device endpoints.	<code>list_activities</code>	Optional <code>start_date</code> , <code>end_date</code> , <code>device_label_1</code> , <code>device_label_2</code>	A list of activities that match the specified filters.
The user requests a list of vulnerabilities, optionally filtered by start date, end date, device label, or minimum CVSS score.	<code>list_vulnerabilities</code>	Optional <code>start_date</code> , <code>end_date</code> , <code>device_label</code> , <code>cvss_score</code>	A list of vulnerabilities that match the specified conditions.
The user requests an explanation of a specific vulnerability.	<code>explain_vulnerability</code>	Required <code>vulnerability_cve</code>	An explanation of the requested vulnerability.
The user requests vulnerabilities related to a specific component.	<code>vulnerability_component</code>	Required <code>component_label</code>	A list of vulnerabilities associated with the specified component.
The user requests vulnerabilities grouped by mitigation strategy.	<code>similar_vulnerabilities</code>	Not applicable	A list of vulnerabilities grouped by mitigation strategy.
The user requests a random anecdote related to a vulnerability.	<code>vulnerability_anecdote</code>	Not applicable	An anecdote related to a randomly selected vulnerability.
The user subscribes to periodic event notifications summarizing the state of the ccv network.	<code>subscribe_events</code>	Optional <code>interval_time</code>	A confirmation message for the subscription.
The user unsubscribes from event notifications.	<code>unsubscribe_events</code>	Not applicable	A confirmation message for the unsubscription.
A user requests disconnection, thereby deauthenticating the user from the chatbot.	<code>disconnect</code>	Not applicable	A confirmation message for disconnection.
The chatbot fails to understand the user's request.	<code>unknown</code> (fallback intent)	Not applicable	A message indicating that the request was not understood.

# Chapter 5

## System Architecture Overview

The remainder of this thesis follows a component-driven structure, with each chapter focusing on a specific part of the designed and implemented solution. Each component is examined in detail, culminating in a demonstration of how they work together to deliver user responses. Before delving into these components, this chapter provides a high-level overview of the system illustrated in [Figure 5.1](#).

While this architectural diagram provides an overview of how the chatbot operates, it is important to note that the components do not function as independent microservices. Rather, each component represents a logical grouping of objects and functions within a single Python application. The entire system runs as one unified application, with these components collaborating internally to perform their respective tasks. In essence, the diagram conceptually illustrates the application's architecture, rather than depicting distinct, standalone services.

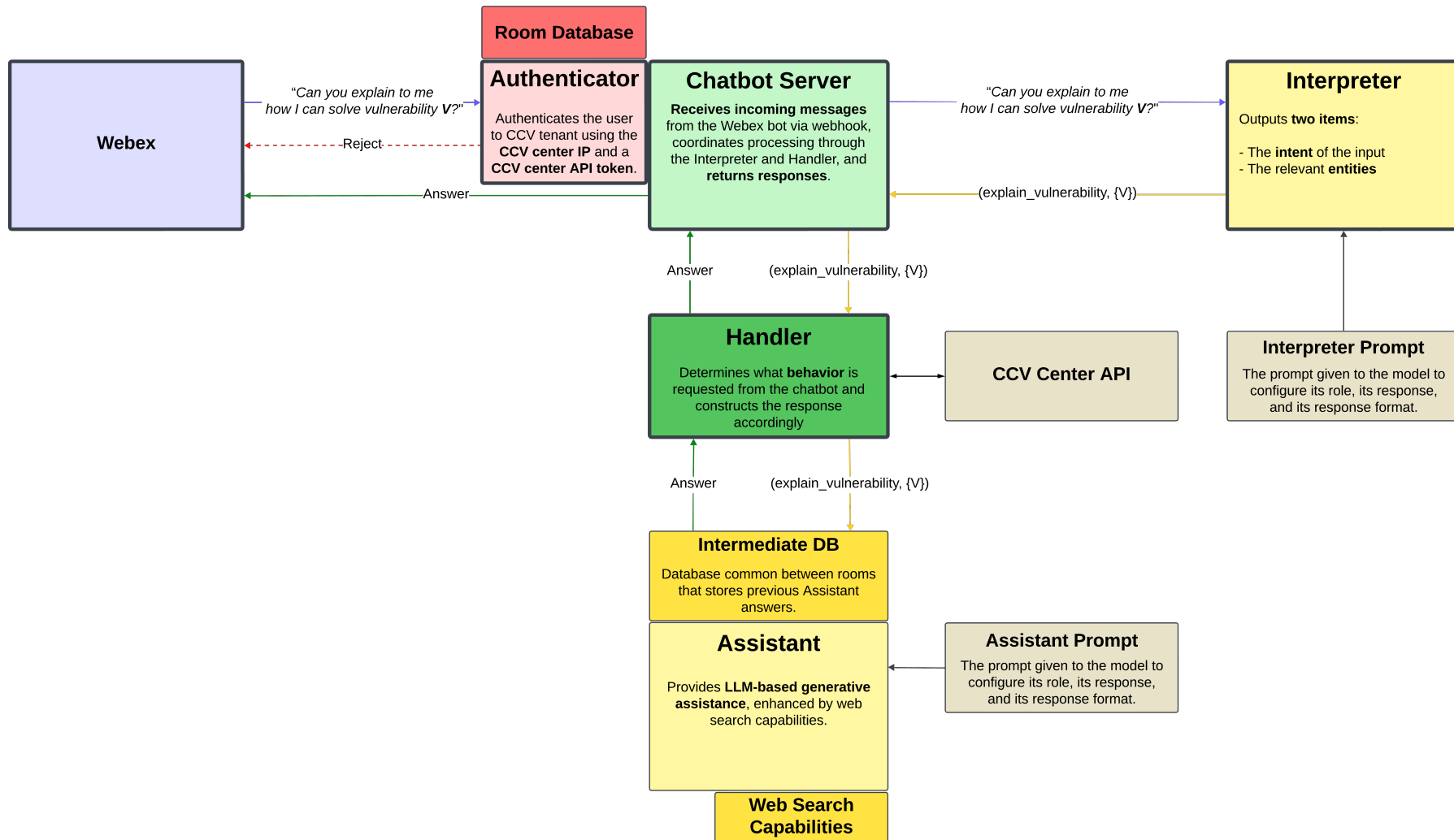


Figure 5.1. High-level system architecture illustrating interactions between the chatbot's components.

## 5.1 Chatbot Server

Configured in Chapter 6, the *Chatbot Server* serves as the central hub for handling messages sent from a Webex user to the bot introduced in Section 2.4.1. It orchestrates the execution of the components responsible for processing the logic between message reception and response delivery, and returns the final response to the originating Webex user.

## 5.2 Authenticator

The first component a user interacts with when using the chatbot via Webex is the Authenticator, described in Chapter 7. This component verifies whether the user has access rights to the targeted CCV center. Notably, the Authenticator authenticates a room—a term encompassing both direct messages and spaces (further discussed in the Authenticator chapter)—rather than an individual user.

While the authentication task may appear straightforward, it must be adapted to the metadata available in Webex messages. Additionally, to preserve confidentiality—especially in spaces where messages are visible to all members—the system must employ a secure, privacy-preserving approach.

## 5.3 Interpreter

As explained in Section 3.2, the first critical step is to extract both the intent and the relevant entities from the user’s message using IC and ER. These tasks are handled by the *Interpreter* component, described in Chapters 8 and 9. Initially, the Interpreter is implemented using a conventional ML approach, and is later enhanced through a modern, LLM-based proof of concept.

## 5.4 Handler

Once the intent and entities have been parsed, the *Handler* component—described in Chapter 10—is responsible for executing the appropriate action or forwarding the message to the *Assistant* component when a generative response is required.

## 5.5 Assistant

The LLM-based *Assistant*, described in Chapter 11, is responsible for several specific yet crucial tasks within the project—such as explaining how to address a given vulnerability, interpreting the meaning of a CCV tag, identifying the origin of device vendor, and more. Its primary strength lies in leveraging an LLM to dynamically generate informative, context-aware textual responses.

# Chapter 6

## Chatbot Server

In this chapter, we introduce the *Chatbot Server*—the central component responsible for receiving messages from Webex, coordinating the relevant components to process those messages, and delivering responses to the originating user. We begin by detailing the server initialization process, followed by an explanation of how Webex’s API facilitates communication between the chatbot and the platform.

Although Webex provides numerous API endpoints to support a wide range of functionalities, this chapter focuses specifically on three: receiving messages, sending responses, and creating bot webhooks.

### 6.1 Server Initialization

#### 6.1.1 Web Server Configuration

In this section, we set up a local Python server handling HTTP requests as well as a Reverse Proxy (RP) forwarding HTTPS requests to the former server.

##### Local Web Server with Flask

The chatbot application will receive messages in a format explained in Section 6.2 through a specified port. To implement this functionality, Flask, a lightweight web framework for Python [50], is used to enable the application to listen on HTTP port 8080 on localhost. The ability to receive messages from Webex, which is hosted in the public cloud, is facilitated by the RP, as explained next.

##### Reverse Proxy (RP) with NGINX

NGINX, a high-performance web server and RP [51], is used to listen on HTTPS port 443 and forward incoming requests to local port 8080, where the Flask server is running. The RP serves both as a secure interface between the web server and the internet, and as an HTTPS terminator, offloading Transport Layer Security (TLS) encryption and forwarding requests to the application over HTTP.

### 6.1.2 Webhook Configuration

Webhooks, as introduced in Section 2.4.1, are configured using the Webex Webhook API [47]. Configuration details are provided in Appendix B. Each webhook defines a target URL to which messages are delivered, along with a resource type that specifies the kind of data forwarded. At this stage of the thesis, we focus exclusively on the *Message* resource type, which corresponds to plain text messages sent by Webex users—the primary mode of interaction with the chatbot.

As a result of this step, all messages (as defined by Webex’s Message resource type) sent to the chatbot will be forwarded to the `/message` endpoint of the Flask web server.

## 6.2 Message Reception

Before implementing the receiving logic, we need to parse what metadata is sent to the webhook. Using the newly created Webex bot, which forwards content to the URL specified by the webhook, we send a message and receive a JavaScript Object Notation (JSON)—a data-interchange format—at the server end, as illustrated in Figure 6.1.

```
1 {
2   "id": "YmIy",
3   "name": "message-webhook",
4   "targetUrl": "https://example/message",
5   "resource": "messages",
6   "event": "created",
7   "orgId": "xMGY",
8   "createdBy": "jNWE",
9   "appId": "MWFh",
10  "ownedBy": "creator",
11  "status": "active",
12  "created": "2024-07-24T12:51:05.186Z",
13  "actorId": "xOTg",
14  "data": {
15    "id": "YmZh",
16    "roomId": "ZDdj",
17    "roomType": "direct",
18    "personId": "xOTg",
19    "personEmail": "lorem.ipsum@example.com",
20    "created": "2025-03-12T06:39:04.838Z",
21  },
22 }
```

Figure 6.1. Raw JSON content received at the server after a message is sent to the Webex Bot. This JSON contains metadata related to the webhook but not the actual message content.

The top-level properties in the webhook payload pertain to the webhook itself. While these

can be useful for debugging, they do not contain the actual content of the message sent by Webex. Instead, the relevant information is located within the `data` property.

Within `data`, we can identify metadata such as the user (or person) ID and the room ID. However, the message content itself is not included.

To retrieve the actual message text, a separate call must be made to a Webex API endpoint [52], using the message identifier (provided as the `id` field within `data`) and the bot's bearer access token. This call returns the JSON illustrated in Figure 6.2.

```
1  {  
2    "id": "YmZh",  
3    "roomId": "ZDdj",  
4    "roomType": "direct",  
5    "text": "Hello World!",  
6    "personId": "xOTg",  
7    "personEmail": "lorem.ipsum@example.com",  
8    "created": "2025-03-12T06:39:04.838Z"  
9  }
```

Figure 6.2. Raw JSON content returned by the Webex API when querying the message using the message ID. This JSON includes the actual message content, which is needed for processing.

At last, we receive the message corresponding to the ID and can extract its content: **Hello World!**.

## 6.3 Response Delivery

Once the bot retrieves the message, the chatbot's logic—detailed in subsequent chapters—is executed based on the content of the message. A response is then sent back to the same Webex user from which the original message was received. The process unfolds as follows: the chatbot's response is divided into chunks of up to 5400 characters, which is the maximum allowed per Webex message [53]. Each chunk is then sent sequentially to the Webex user using the appropriate Webex API endpoint.



# Chapter 7

## Authenticator

In this chapter, we cover how authentication with the chatbot is performed, which occurs indirectly through authentication with a CCV center, as only users with access to a center should be able to interact with the chatbot.

This chapter begins by introducing the authentication mechanism provided by the CCV center. It then details how authentication persistence is handled, how sensitive credentials are securely collected using Adaptive Cards, and how these credentials are validated before being stored. Finally, key implementation details, including webhook and server configuration, are presented to illustrate how authentication is integrated into the chatbot system.

Before proceeding with this chapter, it is important to clarify that no assumptions are made regarding the underlying infrastructure on which the chatbot operates, nor the security guarantees of each component involved in the authentication mechanism. This is an intentional omission, as the primary focus of this thesis is the design and implementation of a conversational AI-powered chatbot for CCV. Ensuring that users can authenticate to a CCV center in a user-friendly manner—while avoiding obvious security vulnerabilities—remains within the scope of this work. However, the infrastructure supporting these mechanisms, as well as the protection of server-stored credentials against unauthorized access, falls outside the intended scope of this proof of concept.

### 7.1 CCV Center Authentication Mechanism

CCV provides two services that require authentication. The first is authentication to the web UI of the CCV center using an email address and a password. Once authenticated, administrators gain access to the creation of authentication tokens, which allow the use of the CCV center API endpoints, requiring the token and the IP address of the host on which the CCV center instance is running.

For this work, the second authentication use case is particularly relevant, as the chatbot interacts exclusively with CCV’s API endpoints to retrieve data in response to user messages.

The chatbot’s authenticator will require three key mechanisms:

1. The user must stay authenticated between messages for a given amount of time.

2. Authentication attributes must remain confidential.
3. Authentication attributes must be valid.

Before elaborating on these mechanisms, it is important to clarify that NLU will not be used for the authentication use case. This decision is based on two main reasons:

1. As previously mentioned, NLU is typically used to classify message intent. For instance, if a user sends something like “*I would like to authenticate*”, the chatbot could infer the **authenticate** intent. However, this is unnecessary here: before authentication, all other intents are invalid, making **authenticate** the only permissible one. Since this intent is implicit, performing IC introduces avoidable overhead. Nonetheless, NLU could remain useful for extracting relevant entities via ER—specifically the IP address and access token—which leads to the next point.
2. Using ER to extract credentials such as IP addresses or access tokens requires identifying the exact segments of text containing these values. This is error-prone: users may vary formatting, make typos, or embed credentials in unrelated text. Given that credentials are sensitive and require exact matches, even minor errors can result in failures or security risks. In short, while NLU excels at interpreting natural language, it is not suited for tasks demanding absolute precision like credential extraction.

## 7.2 Authentication Persistence

Keeping users authenticated throughout their experience requires the need to store credentials. These credentials must only be used by the entity that provided them. To ensure this, we must determine the identification element by which the chatbot will recognize each entity<sup>1</sup>.

As shown in Figures 6.1 and 6.2, the two most straightforward options for identifying the entity providing credentials are the **personId** and the **roomId**, both included in the metadata of messages from Webex direct messages and spaces. While both are valid, this chatbot uses the room ID for its convenience: it allows an entire Webex space to authenticate collectively, rather than requiring each user to authenticate individually. Responsibility then falls to the credential owner to decide whether all members of a given space should be granted access to the chatbot—and, by extension, to the CCV center.

As a result, the provision of a single Webex space member’s credentials is sufficient for all members to communicate with the chatbot. Since credentials provide full access to the CCV API and are not attached to a specific role with a defined set of access rights, this does not introduce an Identity and Access Management (IAM) vulnerability.

Every time the chatbot receives a message, it will check if the room already exists in the dedicated room database. If it does, the message will pass through the rest of the system. Otherwise, it will start the authentication process developed in section 7.3.

---

<sup>1</sup>As stated earlier, we do not elaborate on the relevance and security of the database technology used to store credentials, as this is outside the scope of this work. However, this aspect is essential to the identification process of the Webex entity sending the message to the chatbot.

## 7.3 Credential Collection and Confidentiality

In the current setup, when interacting with the chatbot within a Webex space, credentials cannot be kept private from other members. All messages sent to the chatbot are visible in plain text to everyone in the space. To mitigate this issue, users require a means to transmit sensitive credential information directly to the chatbot without exposing it to others. The most effective and elegant solution to this problem is the use of an *Adaptive Card*.

### 7.3.1 Adaptive Card

Based on Microsoft’s Adaptive Card specification, *Adaptive Cards* in Webex provide a way to display interactive content, such as forms or buttons. They are defined in JSON, which describes both the layout and behavior of the card [54].

An Adaptive Card is well-suited for handling authentication. On one hand, it offers users a familiar, form-like interface for entering credentials, similar to login forms found in most applications. On the other hand, it ensures confidentiality: the submitted values are visible only to the user who entered them and to the chatbot, preventing exposure to other users.

As previously explained, authentication is the only valid use case available to a user prior to being authenticated. Once authenticated, multiple use cases—referred to as intents—become accessible, including the `disconnect` intent, which allows for deauthentication.

## 7.4 Integration and Implementation Details

Regardless of the message sent, if the originating room has not been previously registered, an authentication Adaptive Card is sent, as shown in Figure 7.1. This card includes input fields for the CCV center’s IP address and the corresponding CCV API token.

## Authentication

By using this chatbot, you allow it to use your secret API token to interact with the Cisco Cyber Vision API.

Data submitted by a user is encrypted and stored within the Webex platform.

Cyber Vision Center IP \*

X.X.X.X

Cyber Vision API Token \*

Can be found under Admin > API > Token

Submit

Figure 7.1. Webex Adaptive Card used for authentication, created with Webex’s Card Designer tool [55], and delivered to Webex users.

To enable the chatbot to receive submitted credential data from the card, a new webhook must be created, detailed in Appendix B, along with a corresponding `/card` endpoint in the Flask server to handle the incoming data.

The submitted credentials are then verified for validity, as described in Section 7.4.1, before being stored in the room database. At this stage, the database contains entries consisting of the room ID, the CCV center’s IP address, and a valid CCV API token extracted from the submitted Adaptive Card.

### 7.4.1 Credential Validation

Once the user has provided both the CCV Center IP address and the API access token, the validity of these credentials must be verified before populating the database. Since CCV does not offer a RESTful API endpoint specifically designed for credential validation, the next best approach is to call an existing endpoint with minimal side effects. The selected endpoint for this validation is the one that returns the CCV Center’s software version.

The process involves first creating a mock entry in the room database, then calling the selected RESTful API endpoint. Based on the response, the system either finalizes the

database entry or returns a “`Bad credentials`” message. The version endpoint provides clear feedback on which part of the authentication process failed—such as an unreachable IP address or an invalid access token—making it well-suited for this use case and enabling the chatbot to generate targeted, informative responses.

Throughout this thesis, messages from both Webex spaces and direct messages are collectively referred to as user messages. While the chatbot authenticates based on the room ID, this implementation detail concerns only the Authenticator and the Chatbot Server. For all other components that implement conversational logic, the source of the message is functionally equivalent, and the term user message remains a more intuitive abstraction.

## Chapter 8

# Interpreter: Conventional NLU for Intent Classification and Entity Recognition

The Interpreter is responsible for extracting the intent of a message received from a Webex user and the relevant entities it contains. This information enables the Handler component—discussed in Chapter 10—to initiate the appropriate response delivery process. Therefore, the Interpreter is built around the principles of NLU.

To understand the final design of the Interpreter, we divide its development into two implementation approaches. This chapter focuses on the first approach, which relies on conventional ML techniques. The following chapter introduces the second approach, which leverages LLMs and prompt engineering.

Although the first approach employs well-established ML models, the internal mechanics of these models are not the focus of this thesis. Instead, we concentrate on their integration into the chatbot’s NLU pipeline and evaluate their performance in terms of IC and ER.

To illustrate this approach, we begin by applying it to the IoT OD EDM service—a platform with a distinct scope and purpose from CCV. The initial development of the chatbot, and consequently the first version of the Interpreter component, was centered around the IoT OD EDM platform. However, as Cisco began phasing out IoT OD EDM, the focus of development shifted toward the CCV platform, where the chatbot was adapted and extended to align with its new operational context.

Furthermore, because the LLM-based Interpreter explored in Chapter 9 demonstrates promising performance and addresses several important limitations of the current Interpreter—outlined in Section 8.9—the final chatbot implementation adopts this LLM-based approach. As a result, the conventional ML approach presented in this chapter has not been evaluated against the CCV use cases defined in Chapter 4.

To contextualize the Interpreter’s development, this chapter first provides a brief overview of IoT OD EDM (Section 8.1).

We then define the intents and entities the Interpreter must recognize in Section 8.2, which guides both system design and evaluation.

Following this, we explore the fundamentals of the method and the available tools for implementing NLU, as discussed in Section 8.3.

Once this foundation is established, we present the implementation of the solution in the context of the IoT OD EDM use cases. To that end, we:

- Present the domain-intent-entity hierarchy in Section 8.4, which decomposes tasks into subtasks to be handled independently.
- Explain how the data used to train and test the models is collected and prepared in Sections 8.5 and 8.6.
- Explore the ML models used to perform the various NLU tasks in Section 8.7, and evaluate their performance in Section 8.8. Multiple models are assessed to determine which are best suited for each task.
- Conclude with a discussion of the advantages and limitations of the approach in Section 8.9, followed by a practical summary of the chapter in Section 8.10.

## 8.1 Cisco IoT Operations Dashboard (IoT OD)

Cisco’s IoT Operations Dashboard (IoT OD) is a cloud-based IoT services platform designed to securely connect and manage industrial network devices such as routers, switches, and gateways. In addition to maintaining connectivity, the platform collects and manages data from these devices and provides a comprehensive view of connected industrial assets, including sensors and actuators [56]. While IoT OD offers multiple services for various use cases, the service relevant to this work is the Edge Device Manager (EDM).

### 8.1.1 Edge Device Manager (EDM)

EDM is a Cisco service that enables the provisioning, management, and monitoring of IoT network devices. Provisioning refers to setting up and configuring devices to prepare them for operation within an IoT network. Once provisioned, the devices are continuously monitored to track their performance, resource usage, and health status. For the purpose of defining intents, the following (non-exhaustive) EDM functionalities are considered:

- **Inventory (devices):** Listing and managing network devices. Each device includes attributes such as a unique identifier, installed Cisco IOx applications, geographical location, resource usage, configuration details, and more.
- **Applications:** Listing and managing Cisco IOx applications running on the tracked network devices.
- **Configuration:** Listing and managing configuration groups, which allow administrators to apply shared configuration templates across devices of the same type.
- **Operations:** Listing and managing alerts, which indicate abnormal activity, and events, which report significant network activity.

Note that the description above provides a simplified overview of IoT OD EDM, as a detailed understanding of the platform is not required for the design and implementation of the Interpreter’s first approach. Additionally, while authentication with the IoT OD

EDM RESTful API is necessary for enabling integration between the chatbot and IoT OD EDM, this aspect is considered out of scope, since the final implementation targets the CCV platform instead.

## 8.2 Intent and Entity Definition

Some IoT OD EDM functionalities are exposed through API endpoints. While these endpoints will not be described in detail, they provide access to the functionalities introduced in Section 8.1.1.

Although the use cases covered are relatively simple, they are sufficient to assess the strengths and limitations of this conventional NLU. For this purpose, in Table 8.1, we define ten intents and one entity that the IoT OD EDM chatbot must be able to recognize accurately.

Table 8.1. IoT OD EDM chatbot use cases with corresponding intents, entities, and example user messages.

Use Case	Intent	Entity	Example Message
A user greets the chatbot	<code>greet</code>	Not applicable	Hello!
A user ends the conversation	<code>exit</code>	Not applicable	Goodbye!
A user requests help on available functions	<code>help</code>	Not applicable	What can you help me with?
A user requests all configuration groups	<code>get_all_configuration_groups</code>	Not applicable	What configuration groups are available?
A user requests a list of all alerts	<code>get_all_alerts</code>	Not applicable	List all alerts.
A user requests a list of all events	<code>get_all_events</code>	Not applicable	List all events.
A user requests a list of all devices in the network	<code>get_all_devices</code>	Not applicable	List all devices.
A user requests a list of all device locations	<code>get_all_device_locations</code>	Not applicable	List all device locations.
A user requests a list of all device applications	<code>get_all_device_applications</code>	Not applicable	List all device applications.
A user requests information about a specific device by name	<code>get_device_by_name</code>	<code>devicename</code>	Give details on device X.

Unlike the scope defined for CCV in Chapter 4, no **unknown** intent has been specified. Whether or not to include such an intent depends on the desired behavior of the chatbot: should it always assign a message to one of the known intents, or should it be able to flag vague or irrelevant messages to the user—which, in many cases, is more user-friendly. In this case, since the final scope of the chatbot is limited to CCV rather than IoT OD EDM, and because including an **unknown** intent does not affect the implementation process described in this chapter, it was decided to omit it.

The objective of this chapter is to implement an **Interpreter** component capable of classifying the defined intents and recognizing the `devicename` entity in the context of the `get_device_by_name` intent. Multiple datasets will be used to train models for IC to



evaluate their ability to differentiate between intents. For ER, a single dataset is sufficient, as the goal is not to differentiate between multiple entity types, but to assess whether the correct word in the sentence has been accurately tagged.

In the following sections, we describe various approaches for building a component that performs both IC and ER. While these methods follow similar strategies, they differ in their level of abstraction.

## 8.3 NLU Implementation Strategies

Having established the concepts of IC and ER in Chapter 3, we now examine several strategies for implementing them. We focus on three primary ML-based strategies for building the conventional **Interpreter** component responsible for IC and ER. Each approach presents its own strengths and limitations, which are discussed in the following subsections. Based on this comparison, the most suitable method for the objectives of this project will be selected.

It is important to note that this section does not address rule-based approaches, such as IC through keyword matching or ER via dictionaries. While such techniques can be effective in simple systems, they are generally less flexible and are outperformed by ML-based methods, which are the focus of this chapter.

### 8.3.1 No-Code/Low-Code Cloud NLP Solutions

A common starting point for enabling NLU, and more broadly, NLP, is to leverage cloud-based services tailored for developers with limited expertise in ML or NLP. Popular solutions in this category include Google Dialogflow CX [57], IBM Watson [58], and Amazon Lex [59]. These platforms simplify the development of systems capable, among others functionalities, of IC and ER. The setup is typically straightforward: define example utterances for each intent and, where applicable, annotate entities within those utterances. Once configured, predefined responses can be assigned directly via the web UI, or the service can be integrated into an application via API as a middleware layer [60].

The internal logic used by these services to perform NLU is abstracted away from the developer, with limited options for customization. This lack of transparency and control makes such platforms unsuitable for the current work. While effective for simpler applications, they lack the flexibility required to adapt to the specific needs of more complex use cases—such as the one addressed in this project. In particular, customizing the training pipeline or debugging the model’s behavior is either extremely limited or not possible at all [61, 62].

### 8.3.2 Machine Learning (ML) Toolkits

At the other end of the spectrum are ML toolkits, which provide full flexibility in developing NLP solutions by exposing low-level access to state-of-the-art algorithms. These include traditional models such as Logistic Regression (LR) and Support Vector Machine (SVM) (via libraries like Scikit-learn [63]), as well as modern neural architectures such as Convolutional Neural Networks (CNN), Long Short-Term Memory Networks (LSTM), and Transformer models, implemented using frameworks like TensorFlow [64] and PyTorch [65].

Additionally, high-level libraries such as HuggingFace Transformers [66] simplify access to pretrained transformer models, supporting efficient fine-tuning and deployment for a broad range of NLP tasks.

While this approach offers the flexibility needed to solve most IC and ER problems—including those encountered in this work—it also introduces significant complexity. The next approach balances the advantages of low-level control with the convenience of abstraction, making it more practical for developing production-ready conversational agents [61, 62].

### 8.3.3 Conversational-AI Platforms

Conversational AI platforms represent a middle ground between no-code/low-code services and full ML frameworks. They offer a compelling balance between flexibility and complexity, making them well suited for implementing the conventional Interpreter component in this first approach. These platforms are purpose-built for developing production-grade NLP applications, streamlining everything from data collection and labeling to model training, evaluation, and deployment [61].

Among the most widely used platforms are Rasa [67] and Cisco MindMeld [68]. Both are appropriate for this work, but MindMeld was selected due to its native support for Webex integration and its use as the NLP engine in Cisco Botlite—a no-code/low-code cloudplatform for building NLP solutions. While the Webex integration module has been extensively adapted to meet the specific requirements of this chatbot, it provided a strong starting point for establishing communication between Webex and the chatbot.

### 8.3.4 Cisco MindMeld

Cisco MindMeld is a Python-based conversational AI platform designed for building production-grade dialogue systems. It supports locally trainable and executable ML models to power a complete NLP pipeline, including IC and ER, both of which are used in the implementation of the Interpreter component.

In this initial implementation of the Interpreter, MindMeld is employed to build a component capable of extracting both user intent and relevant entities from natural language input [69]. The following sections provide a step-by-step overview of how MindMeld’s NLU capabilities are applied to classify and recognize the intents and entities defined earlier in Section 8.2.

## 8.4 Domain, Intent, and Entity Hierarchy

The NLU architecture used by MindMeld<sup>1</sup> is organized into four layers.

The top Domain Classification (DC) layer categorizes user messages into broad topical areas, or “*domains*”, such as **music**, **sports**, or **device**. Each domain contains a set of related intents and uses the same underlying classification logic as IC. DC is performed before IC to identify the relevant domain, helping to narrow down possible intents.

---

<sup>1</sup>The data classification hierarchy used by MindMeld is common across several conversational platforms and is not unique to MindMeld.

Once the domain is identified, the IC layer assigns the message to a specific intent within that domain. For example, within the `device` domain, intents such as `get_devices` and `get_device_by_name` are valid.

Following domain and intent classification, ER is applied to identify entities within the input message. For instance, for the `get_device_by_name` intent, the device name—`devicename`—is extracted as an entity.

The final layer, Role Classification (RC), adds semantic distinction to identified entities. This is useful when entities of the same type play different roles. For example, both flight departure and arrival times are `time` entities, but they represent different semantic roles: `departure` and `arrival`. However, this fourth layer is not used in this work [70].

While performing DC before IC was initially motivated by the classification strategy used in MindMeld, it offers two key advantages over performing IC directly [71]:

1. **Modularity:** Classifying a message by domain, first, makes the system easier to extend, particularly when adding new use cases. Each domain can maintain its own set of intents and entities, supporting a cleaner separation of logic.
2. **Scalability:** DC narrows down the set of candidate intents, reducing the complexity of IC. This improves classification efficiency and accuracy, especially as the number of intents grows.

However, this approach also introduces a dependency: if DC fails, IC will fail as well. This underscores the importance of defining well-separated and unambiguous domains.

Given this hierarchy, the intents defined in Section 8.2 are organized as shown in Figure 8.1.

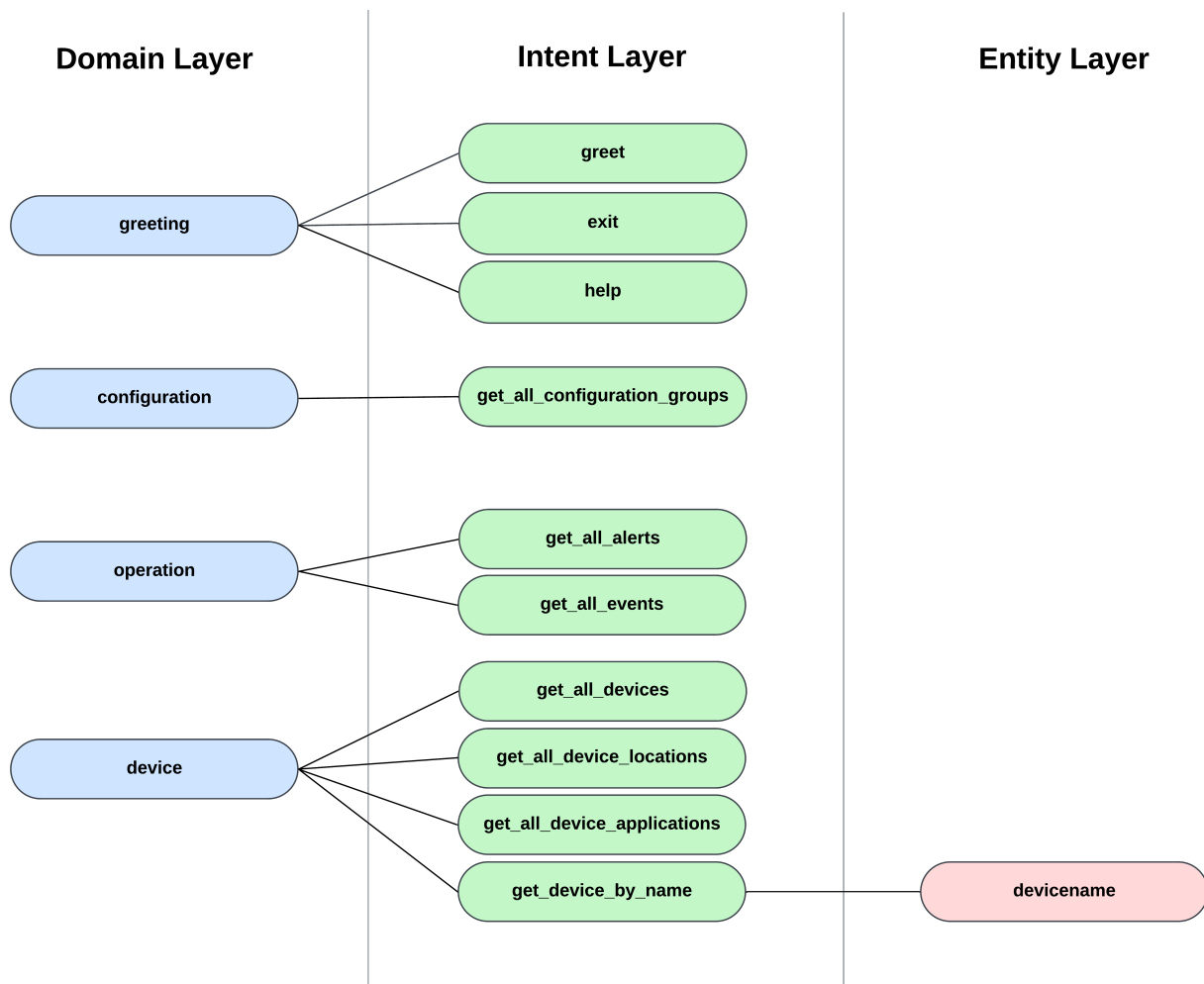


Figure 8.1. Defined intents and entities organized according to the domain-intent-entity hierarchy.

With the data hierarchy defined, the next step is to collect and prepare training data—specifically, user utterances for each defined intent, along with any relevant entities annotated within those utterances.

## 8.5 Data Collection

### 8.5.1 Synthetic Data Generation

The input data that the ML models must classify consists of user utterances—textual messages received from Webex users. Therefore, the first step is to collect representative utterances for each intent.

Since this project is novel, no prior data has been collected. As a result, two general approaches can be considered for data collection:

1. Generating data synthetically.
2. Gathering data from actual real examples.

While the latter would yield more representative and naturally phrased data, it was not

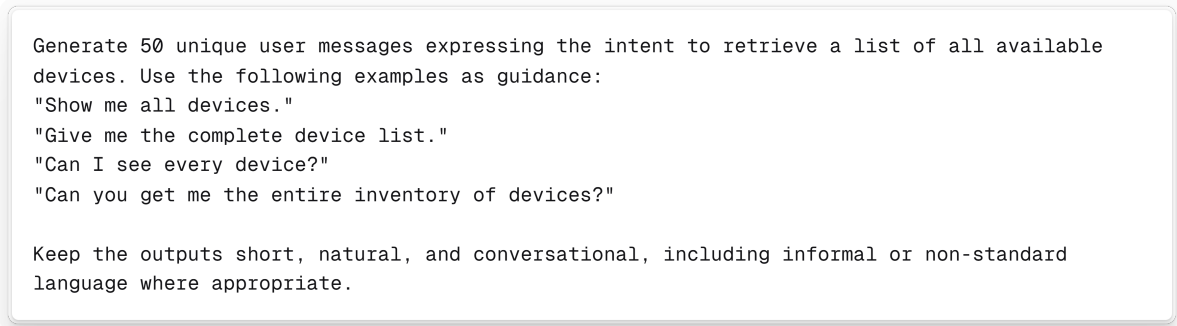
feasible in this work due to logistical constraints. Consequently, synthetic data generation was adopted to simulate realistic utterances.

Synthetic data was produced through a combination of manual creation and LLM data augmentation. This hybrid approach was chosen for its ability to generate high-quality, human-like utterances, particularly when employing few-shot prompting—that is, providing the LLM with examples of expected utterances to guide generation.

The prompts used to guide the LLM included:

- Example utterances corresponding to the target intent.
- Instructions to generate  $N$  unique phrases.
- A preference for short, natural, and conversational expressions (including informal or non-standard language).

Figure 8.2 illustrates a sample prompt created for demonstration purposes. It does not represent the actual prompts used in data generation, as those were iteratively refined and manually reviewed throughout the data creation process.



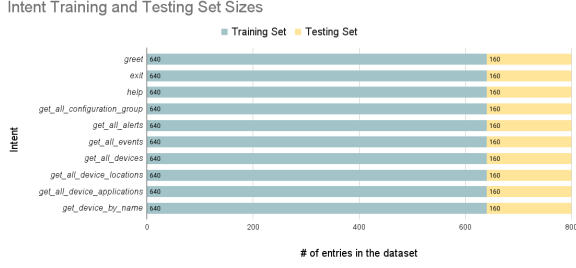
```
Generate 50 unique user messages expressing the intent to retrieve a list of all available devices. Use the following examples as guidance:  
"Show me all devices."  
"Give me the complete device list."  
"Can I see every device?"  
"Can you get me the entire inventory of devices?"  
  
Keep the outputs short, natural, and conversational, including informal or non-standard language where appropriate.
```

Figure 8.2. Illustrative example of a prompt used for synthetic data generation.

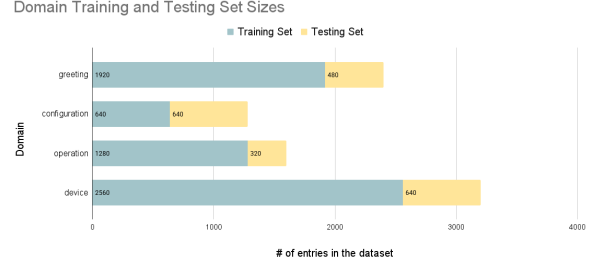
### 8.5.2 Dataset Balancing and Splitting

To avoid dataset imbalance and reduce the risk of biased intent prediction during training [72], the same number of examples,  $N$ , was generated for each intent. However, since DC models are trained on the combined examples from all intents within a domain, achieving perfect balance at the domain level becomes challenging when domains contain differing numbers of intents. As a result, some domains contribute more training data than others, which can negatively impact the performance of models that are more sensitive to such imbalances, as will be seen in Section 8.8.2.

Figures 8.3a and 8.3b illustrate the sizes of the intent and domain datasets, respectively. Each dataset is split into training and testing subsets using an 80/20 ratio, as preparation for performance evaluation.



(a) Intent classification.



(b) Domain classification.

Figure 8.3. Training and testing dataset sizes for intent and domain classification.

### 8.5.3 Data Organization and Annotation with MindMeld

MindMeld simplifies the process of organizing collected data into a labeled dataset by associating each utterance with its corresponding intent and, where applicable, annotated entities. This is achieved through a specific folder-based structure aligned with the domain-intent-entity hierarchy illustrated in Figure 8.1.

In this structure:

- Each domain is represented as a top-level folder.
- Within each domain folder, subfolders correspond to individual intents and contain the relevant training data.
- Inside each intent folder, a file named `train.txt` stores example user utterances.

If an intent requires ER, entities are annotated directly within the training utterances using MindMeld’s inline annotation syntax.

For example, the domain folder `greeting` contains the intent folders `greet`, `exit`, and `help`, each with its own `train.txt` file containing utterance examples. Similarly, the domain `device` includes the intents `get_all_devices`, `get_all_device_locations`, `get_all_device_applications`, and `get_device_by_name`, each with a corresponding `train.txt` file.

Each `train.txt` file contains representative utterance examples relevant to the intent. For instance, the file for the `get_all_devices` intent might include an entry such as: `Can I get an overview of all devices?`, along with other variations.

When intents involve ER, entities are annotated inline using MindMeld’s syntax. Specifically, in the training file for the `get_device_by_name` intent, a sample entry may look like: `Give me information about device {device-123|devicename}`. Here, `device-123` is the instance of the entity, and `devicename` is the entity type to be recognized by MindMeld’s ER system.

Note that, if we were to include the `unknown` intent, the recommended approach would be to construct training data using unrelated utterances to the other intent datasets, as demonstrated in MindMeld’s Home Assistant example blueprint [73].

### 8.5.4 Non-Standard Language Challenges

To develop an effective NLU module, it is essential that the training data reflects how humans communicate through textual messaging. This includes not only grammatically correct language but also non-standard forms such as abbreviations (e.g., *thx*), acronyms (e.g., *FYT*), contractions (e.g., *gonna*), and expressions of emotion (e.g., *hahah*).

Research indicates that these linguistic variations are not random errors but are often deliberate choices used to convey tone or improve typing speed. For instance, studies on instant messaging have found that most such language use is intentional, while genuine spelling mistakes are relatively rare [74].

Excluding these informal elements from the training data can reduce a model’s ability to accurately interpret intents. However, incorporating non-standard language into synthetically generated datasets is both tedious and, to some extent, infeasible without leveraging real user-generated data. In this section, the LLM was prompted to include non-standard terms, but its ability to accurately mimic human-like informal language based solely on instruction prompts remains limited.

While some of the studied ML models in this chapter include models able to robustly tackle this issue (i.e., Bidirectional Encoder Representations from Transformers (BERT) 8.7.1), this limitation in training data quality and accuracy partly motivates the adoption of the second Interpreter component using an LLM-based approach—discussed in Chapter 9.

Before concluding this section, we address two important considerations related to data realism and ER: the use of non-standard language and the applicability of gazetteers.

### 8.5.5 Limitations of Gazetteers and Pattern Matching

A gazetteer is a structured list of predefined entity values, commonly used to guide the predictions of NLU entity recognizers during inference. Instead of contributing to model training directly, gazetteers are applied at runtime to highlight known entities, thereby boosting recognition accuracy for predictable and closed-class categories such as locations, organizations, colors, or standard product types.

However, gazetteers are not suitable for recognizing device names. Device names represent a custom entity type—specifically, an open-class, non-named entity category where values are unconstrained and highly variable. Although the space of possible `devicename` values is technically finite, it is extremely large due to the combination of alphanumeric characters and select special characters (`.`, `+`, `-`), with lengths ranging from 3 to 50 characters. This vast and unpredictable variety makes it infeasible to enumerate all valid forms in a gazetteer, rendering the approach impractical for this use case [75].

Pattern matching is similarly ineffective. Unlike structured entities (e.g., dates or postal codes), device names lack consistent syntactic patterns that can be reliably captured with regular expressions or rule-based logic.

Given these limitations, neither gazetteers nor pattern-based techniques can adequately support the recognition of device names. Instead, the model must rely solely on contextual and embedding-based signals to identify such entities, which may affect recognition performance.

## 8.6 Data Preparation

Once the data has been collected and adequately labeled, it must be prepared for further processing. To that end, each entry passes through MindMeld’s text preparation pipeline, which performs the following operations:

1. **Tokenization:** This step breaks the input text into smaller units called tokens. Tokenization is essential because machine learning models operate on numerical inputs, and embeddings—which convert text into vectors—work on individual tokens. In this work, tokenization is based on whitespace, meaning that each token corresponds directly to a word separated by a space in the original entry.
2. **Normalization:** Normalization standardizes the text to ensure consistency. It includes operations such as trimming whitespace, converting text to lowercase, removing redundant apostrophes, and eliminating or replacing special characters. In this work, we apply MindMeld’s default set of 12 regex-based normalization rules, adapted to preserve the special characters (+, -, .) that may appear in device names. This step enhances both embedding quality and downstream model performance.
3. **Stemming:** Stemming reduces words to their base or root forms, helping to decrease vocabulary size and group semantically similar terms together. MindMeld uses the `EnglishNLTKStemmer`, a simplified version of the `PorterStemmer` from the NLTK library [76], which retains more of the original word structure by removing only inflectional suffixes.

After the data passes through this preparation pipeline, it is ready to be used for training and inference in the NLU models [77].

For example, the original input:

```
└─List all devices I HAVE in my network└─.
```

is processed into the following tokens:

```
list, all, device, i, have, in, my, network2.
```

It is important to note that, before being passed to the ML models, the data undergoes this preparation pipeline—except for BERT-based models, which use a distinct tokenization process, as mentioned in Section 8.7.1.

## 8.7 Machine Learning Models

Selecting appropriate models for DC, IC, and ER is essential to achieving strong NLU performance. MindMeld provides access to a range of traditional ML models—including LR, SVM, Decision Tree (DT), Random Forest (RF), Maximum Entropy Markov Model (MEMM), and Conditional Random Field (CRF) [78, 79]—as well as deep learning (DL) models such as CNN, LSTM, and Transformer-based architectures like BERT [80].

Model selection and configuration for each layer of the NLU pipeline is managed through the `config.py` file, which MindMeld uses to set up, train, and apply models.

---

<sup>2</sup>Note that *devices* becomes *device* rather than *devic*, which would be the case with the standard `PorterStemmer`. This is due to the `EnglishNLTKStemmer`’s more conservative stemming rules, designed to preserve word meaning.



While the focus here is limited to models integrated within the MindMeld platform, these include state-of-the-art approaches for NLU tasks. Although other models exist beyond MindMeld’s scope, the ones provided offer sufficient flexibility and performance for implementing and evaluating the first Interpreter approach presented in this work.

It is important to note that there is no universal or one-size-fits-all model in ML. Performance depends heavily on the task, the nature and size of the training data, and the tuning of model hyperparameters.

Additionally, DC/IC and ER tasks typically rely on different techniques, which are considered separately.

To identify the most effective model for each layer of the domain-intent-entity hierarchy, model selection is approached from two dimensions:

1. Without diving into internal architectures, we provide a high-level overview of how selected models contribute to generating DC, IC, and ER outputs.
2. We then evaluate model performance across standard classification metrics.

### 8.7.1 Domain and Intent Classification

The classification process differs depending on the type of ML model employed. In this structured evaluation, we begin with baseline models based on traditional ML techniques. We then examine a more complex transformer-based model—specifically RoBERTa—and analyze the advantages it offers, despite its increased computational requirements, larger model size, and longer training time.

This thesis does not delve into the internal mechanics of the models for two main reasons: first, these algorithms are already extensively documented in the literature; and second, their technical details are not directly relevant to the objectives of this work. The primary focus is on how each model is applied within the NLU pipeline and how it performs in context.

Although DC and IC are distinct tasks, the process is nearly identical, differing only in the classification target. As a result, the discussion in this section centers on IC, with the understanding that the same methodology applies equivalently to DC—except where performance is evaluated separately.

#### Baseline Models: Traditional Machine Learning Approaches

The first ML models we introduce are LR, SVM, DT, and RF.

As described in Section 8.6, we preprocess each text entry in the training data by generating a sequence of tokens. For example, the input sentences:

1. List all devices I have in my network.
2. Get me all apps from all devices.

respectively become:

1. *[list, all, device, i, have, in, my, network]*
2. *[get, me, all, app, from, all, device]*

Before these tokens can be used as input to a ML model, they must be embedded or transformed into a numerical representation. MindMeld provides several configurations for adapting textual input into feature vectors, including:

- ***n*-grams.** We keep unigrams and bigrams so the model sees individual words *and* their immediate neighbour pairs—just enough local context without a huge feature explosion.
- **Frequency bins (5).** All in-vocabulary tokens are ranked by corpus frequency and sliced into five equal groups (bin 0 = rarest ... bin 4 = most common).
  - For each utterance, we record the fraction of its tokens that fall in each bin.
  - Out-of-Vocabulary (OOV) words are counted separately as an *OOV* feature.

The following table illustrates, as an example, how the embeddings would be constructed for the two sample utterances above.

Table 8.2. Truncated snapshot of the combined feature values for the two utterances. Values were produced using Mindmeld’s `view_extracted_features` function [81].

Feature	Vector 1	Vector 2
<i>Unigram counts</i>		
bag_of_words length:1 ngram:all	1	2
bag_of_words length:1 ngram:apps	0	1
bag_of_words length:1 ngram:devices	1	1
bag_of_words length:1 ngram:from	0	1
bag_of_words length:1 ngram:get	0	1
...		
<i>Bigram counts</i>		
bag_of_words length:2 ngram:all apps	0	1
bag_of_words length:2 ngram:all devices	1	1
bag_of_words length:2 ngram:apps from	0	1
bag_of_words length:2 ngram:devices i	1	0
bag_of_words length:2 ngram:from all	0	1
...		
<i>Frequency-bin fractions</i>		
in_vocab:IV   freq_bin:0	0.1250	0.2857
in_vocab:IV   freq_bin:1	0.3231	0.1429
in_vocab:IV   freq_bin:2	0	0.2264
in_vocab:IV   freq_bin:3	0.1250	0.1429
in_vocab:IV   freq_bin:4	0.1250	0

During both *training* and *inference*, every utterance must first be converted into a composite feature vector like the one described above; the classifiers themselves never operate on raw text. Once this embedding step is complete, the pipeline proceeds as follows:

After converting each utterance into a numerical feature vector, it is passed to one of several classifiers—LR, SVM, DT, or RF.

Each model learns to associate particular feature patterns with one of the intent labels defined in the dataset. During training, the model builds internal representations based

on the labeled examples. At inference time, it applies this learned mapping to predict the most probable intent for new, unseen utterances.

Although these classifiers differ in their underlying algorithms, MindMeld abstracts away those details. One simply specifies the model in the configuration file, defines the parameters and hyperparameters, and MindMeld uniformly manages both the training loop and the inference process.

At inference time, incoming user utterances are tokenized and embedded in the same way as during training. The resulting feature vector is passed through the trained classifier to produce a probability distribution over all known intents. The intent with the highest probability is returned.

### **Transformer-Based Model: BERT-Style Encoder**

In addition to traditional classifiers, we evaluate a transformer-based model utilizing a BERT-style encoder. Unlike traditional models (e.g., LR, DT, RF) that rely on manually engineered features such as  $n$ -gram counts, BERT learns rich, contextual representations directly from raw text.

BERT employs a transformer architecture characterized by self-attention mechanisms, which effectively capture long-range contextual relationships within sequences. Self-attention allows the model to weigh the significance of each word relative to others, enhancing semantic and contextual representations. Crucially, BERT’s architecture is encoder-only and bidirectional, enabling it to consider context from both directions simultaneously.

BERT undergoes extensive pretraining on large-scale corpora using Masked Language Modeling (MLM), which involves predicting masked tokens from their surrounding context, and Next Sentence Prediction (NSP), which assesses sentence continuity. MLM promotes deep contextual understanding, as the model must infer missing information from context.

To fine-tune BERT for IC, raw input text is tokenized into subword units using WordPiece tokenization and augmented with positional embeddings. A special token, [CLS], is prepended to each sequence, acting as a summary representation for the entire input. For example, `List all devices I have in my network.` becomes `[CLS] List all devices I have in my network [SEP]`, with [SEP] marking the sequence boundary.

The model processes the tokenized input through multiple transformer layers, allowing self-attention to capture interdependencies among tokens. The final hidden representation of the [CLS] token, summarizing the entire input sequence, is passed through a classification layer, outputting the predicted intent [82–84].

During fine-tuning, the entire model is trained end-to-end using cross-entropy loss, adjusting both encoder and classification layers simultaneously. At inference time, the model predicts the intent corresponding to the highest probability.

This architecture provides superior language understanding compared to traditional methods. Its bidirectional context modeling captures nuanced phrasing variations, improving generalization. Moreover, pretraining on large corpora allows effective fine-tuning even with limited task-specific data.

For our classification task, we employ RoBERTa, a variant of BERT that removes the NSP objective and replaces WordPiece tokenization with Byte Pair Encoding (BPE). These modifications, combined with other training improvements, have been shown to enhance performance across a range of NLU tasks, including IC [85].<sup>3</sup>

### 8.7.2 Entity Recognition

The ER process runs after DC and IC, and is performed using MindMeld’s entity recognizer, which is a sequence labeling model, also known as a tagging model, that identifies all relevant entities in a given utterance and labels them [79]. Unlike classification models, which are trained across intents, the entity recognizer is trained separately for each intent using intent-specific data.

MindMeld offers the ability to automatically detect system entities such as numbers, email addresses, URLs, and more. However, this feature is not used in our work, as our use cases do not require it. Instead, the only entity that must be recognized is the `devicename` entity from the `get_device_by_name` intent.

Furthermore, as discussed in Section 8.5.5, the use of a gazetteer was ruled out due to the highly variable and open-ended nature of device names.

Among the models provided by MindMeld, we will evaluate the following strategies: MEMM, LSTM, Character-Level CNN Followed by Word-Level LSTM (CNN-LSTM), and BERT.

As for classification, we will not dive into the algorithms behind each model but rather discuss how they are set up for ER.

#### Entity Tagging with IOB/IOBES Labeling

The role of the entity recognizer is to assign a tag to each token in the input utterance, indicating whether it is part of an entity and, if so, its position within that entity. Several standard annotation schemes are used for this purpose, including *IO*, *IOB*, *IOE*, and *IOBES*. Below is a brief overview of these primary schemes:

- **IO** (Inside-Outside): The simplest scheme, where tokens inside an entity are labeled with *I* (Inside), and all other tokens are labeled *O* (Outside).
- **IOB** (Inside-Outside-Begin): Extends IO by marking the beginning of an entity with *B* (Begin), while subsequent tokens in the same entity are labeled *I*. Useful for identifying multi-token entities.
- **IOE** (Inside-Outside-End): Similar to IOB, but it marks the end of an entity with *E* (End) instead of the beginning.
- **IOBES** (Inside-Outside-Begin-End-Single): Combines the strengths of IOB and IOE. It uses *O* for non-entity tokens, *B* for the beginning of a multi-token entity, *I* for internal tokens, *E* for the end, and *S* for entities consisting of a single token.

While the simpler *IO* scheme might suffice for entity types like `devicename`, which always consists of a single token, this work adopts the more expressive Inside-Outside-Begin-

---

<sup>3</sup>ER is instead evaluated using BERT due to compatibility issues with RoBERTa in the MindMeld framework.

End-Single (IOBES) scheme. The goal is to support a wider range of entity types and lengths, as `devicename` serves only as an illustrative example. Prior research has shown that IOBES generally yields superior performance across diverse entity recognition tasks [86, 87].

### Classic Feature-Based Labeling: MEMM

The MEMM is a statistical, feature-based model commonly used for sequence labeling tasks such as ER. Like traditional classifiers such as LR or DT discussed earlier for classification, MEMM operates on tokenized input, representing each token as a feature vector.

A typical feature set includes *n-grams* extracted from the local context of each token. For example:

- **Unigrams** are collected from a window spanning two tokens before to two tokens after the current token (positions  $[-2, -1, 0, 1, 2]$ ).
- **Bigrams** are extracted starting at one token before, the current token, and one token after (positions  $[-1, 0, 1]$ ).

These features help the model capture local lexical patterns, allowing it to make labeling decisions based on both the token itself and its immediate context.

For instance, for the input `Could you update me about {X|devicename}?`, the model identifies “X” as a likely entity based on the contextual features derived from its surrounding tokens.

Despite its strengths in modeling local dependencies, MEMM has limitations. It struggles to capture long-range relationships and complex syntactic structures. Moreover, because it normalizes predictions at each step independently, it is prone to the *label bias problem*, where transitions with fewer possible next labels are disproportionately favored [79, 88].

### Neural Sequence Labeling: LSTM and CNN-LSTM

A foundational neural approach to sequence labeling is the use of LSTM networks—specifically, Bidirectional LSTM (BiLSTM), which extend Recurrent Neural Networks (RNNs). BiLSTMs capture both local and long-range dependencies by processing the input sequence in both forward (left-to-right) and backward (right-to-left) directions, providing richer contextual representations for each token.

In this architecture, input sentences are first embedded using Global Vectors for Word Representation (GloVe) word vectors—pretrained embeddings learned from large, general-domain corpora. These embeddings encode semantic relationships between words, offering meaningful representations even for inputs that may be infrequent in the training data. This is especially valuable for recognizing open-class entities like `devicename`, where training data coverage may be sparse.

To handle OOV words and improve robustness to irregular or novel forms, character-level embeddings are also introduced. These embeddings are generated using a CNN network, which captures morphological patterns within tokens by analyzing subword structures. The resulting character embeddings are concatenated with the word-level GloVe embeddings and then passed through the LSTM network for contextual encoding.

In both the LSTM and CNN-LSTM architectures, a CRF layer is applied on top of the contextualized token embeddings. The CRF models the dependencies between output labels, ensuring coherent and globally optimal tag sequences across the sentence [80, 89].

### Deep Contextual Labeling: BERT

Unlike LSTM and CNN-LSTM architectures, which rely on separately pre-trained word embeddings and sequential modeling, transformer-based models such as BERT provide deep contextual embeddings in a fully end-to-end manner. As discussed previously in Section 8.7.1, BERT eliminates the need for manual feature engineering by capturing rich, bidirectional context through self-attention mechanisms.

For ER, BERT processes the input using WordPiece tokenization and generates contextualized representations for each subword token. These representations are then passed through a CRF layer, which predicts the most likely sequence of entity tags across the sentence, ensuring globally coherent label assignments. The entire model, including the CRF layer, is fine-tuned jointly on the ER task.

## 8.8 Performance Evaluation

### 8.8.1 Evaluation Methodology

As described in Section 8.5, test datasets were created for evaluation purposes. For DC and IC, a Python script was developed to compute standard classification metrics—accuracy, precision, recall, F1-score, and support—using scikit-learn’s `classification_report` function. For ER, MindMeld’s built-in `evaluate` method is used [79], which reports both token-level and sequence-level statistics for entity recognition.

In all cases, precision, recall, and F1-score are reported as *support-weighted averages* to account for class imbalance. This means that each class contributes to the overall metric in proportion to the number of true instances (support) it has in the test set. These metrics are computed in a one-vs-all fashion for each class individually, and the final average is calculated using the following formula:

$$\text{Metric}_{\text{weighted}} = \frac{\sum_{i=1}^N \text{support}_i \times \text{metric}_i}{\sum_{i=1}^N \text{support}_i}$$

where:

- $N$  is the number of classes,
- $\text{metric}_i$  is the precision, recall, or F1-score for class  $i$ ,
- $\text{support}_i$  is the number of true instances of class  $i$ .

This approach is particularly important for DC, where label frequency varies significantly across domains.

The metrics are interpreted as follows:

- **Accuracy:** Proportion of correct predictions among all predictions.

- **Precision:** Proportion of correctly predicted positives among all predicted positives.
- **Recall:** Proportion of actual positives that were correctly predicted.
- **F1-score:** Harmonic mean of precision and recall.
- **Support:** Number of true instances of each class in the test set.

The evaluation results are interpreted with the following assumptions:

- Each model is configured with the best-performing hyperparameters identified during development via grid search.
- Results are specific to the experimental setup described in Sections 8.5 and 8.6. Since the training data is synthetically generated, it may not fully reflect real-world language variability despite efforts to approximate it.

## 8.8.2 Domain and Intent Classification Results

### Independent Evaluation of Domain and Intent Classifiers

In a realistic setup, the performance of the intent classifier would naturally be influenced by the accuracy of the domain classifier. If the domain is misclassified, the intent will certainly be incorrect as well, since IC is domain-specific. However, the goal of this section is to evaluate DC and IC independently, isolating the performance of the intent classifier from any errors introduced by domain misclassification. To achieve this, the evaluation follows a two-step strategy: first, the models responsible for DC are evaluated and their results reported. Then, in a separate evaluation loop, IC models are assessed using the entire test set of utterances, regardless of how the domain classifier performed. This ensures that IC results are not influenced by DC accuracy. As a result, the number of evaluated samples (i.e., the support) for each intent remains consistent with the original test dataset statistics reported in Section 8.5, allowing for fair and equal comparison across all models.

As shown in Figure 8.4, RoBERTa achieves the highest F1-score for DC, reaching 0.9994 followed closely by LR with 0.9775 and SVM with 0.9508. RF and DT, perform decreasingly poorly with 0.8385 and 0.6518, respectively.

However, when computational efficiency is taken into account, RoBERTa performs proportionally worse than LR. Despite their comparable classification performance, RoBERTa’s significantly higher training and inference times, along with greater resource demands, may make LR a more practical choice for DC in scenarios involving large training datasets or constrained computational resources.

A notable limitation in the current setup is that the number of utterances per domain is indirectly determined by the number of intents within that domain, since each intent contributes an equal amount of training data. This creates an imbalance in domain-level support, which disproportionately affects simpler models that are more sensitive to such disparities. While one way to mitigate this would be to standardize the number of intents per domain or increase the number of utterances for domains with fewer intents, both approaches are impractical.



Average Accuracy, Precision, Recall, and F1-Score for Domain Classification

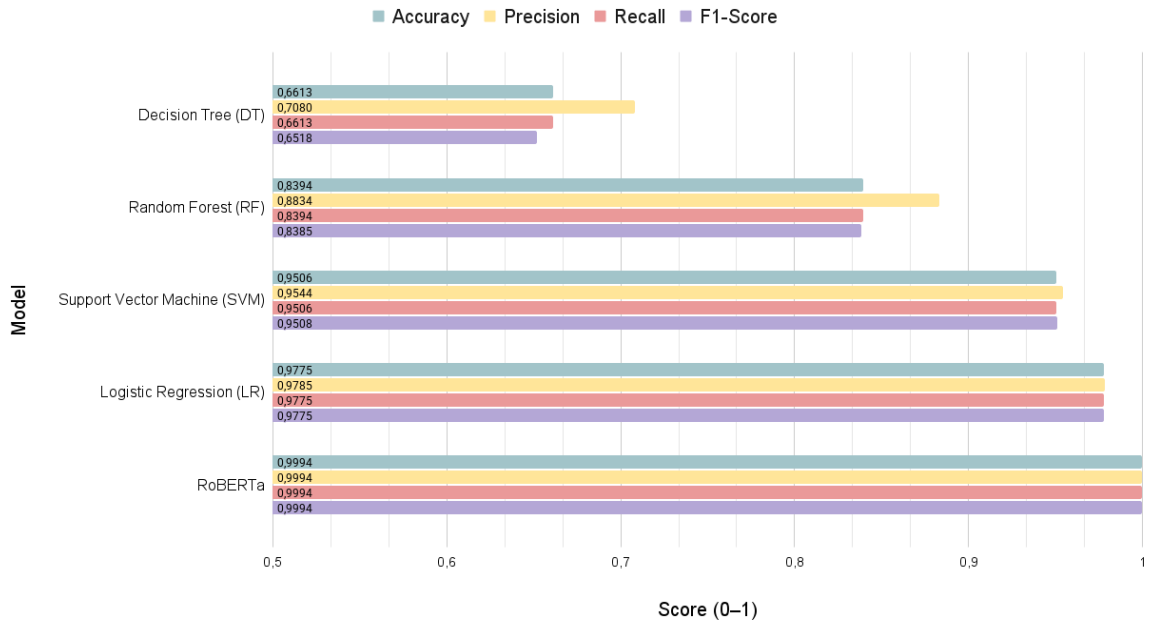


Figure 8.4. Bar chart showing average accuracy, precision, recall, and F1-score for each tested model on DC.

This impact of imbalanced training data across classes is further evident in the case of IC. As shown in Table 8.3, although the overall performance rankings of the models remain consistent—with RoBERTa and LR leading—a model such as RF demonstrates notably improved performance when trained on balanced datasets. This highlights its sensitivity to class imbalance and suggests that its underperformance for DC is at least partly due to disparities in training set sizes.



Table 8.3. Performance Comparison of RoBERTa, LR, SVM, RF, and DT Across Intent Classification Tasks

Domain	Model	Accuracy	Precision	Recall	F1-Score	Support
configuration	RoBERTa	1.0000	1.0000	1.0000	1.0000	160
	Logistic Regression (LR)	1.0000	1.0000	1.0000	1.0000	160
	Support Vector Machine (SVM)	1.0000	1.0000	1.0000	1.0000	160
	Random Forest (RF)	1.0000	1.0000	1.0000	1.0000	160
	Decision Tree (DT)	1.0000	1.0000	1.0000	1.0000	160
device	RoBERTa	0.9953	0.9953	0.9953	0.9953	640
	Logistic Regression (LR)	0.9891	0.9891	0.9891	0.9891	640
	Support Vector Machine (SVM)	0.9734	0.9737	0.9047	0.9735	640
	Random Forest (RF)	0.9266	0.9287	0.9266	0.9254	640
	Decision Tree (DT)	0.5422	0.6423	0.5422	0.5333	640
greeting	RoBERTa	0.9854	0.9856	0.9854	0.9854	480
	Logistic Regression (LR)	0.9438	0.9456	0.9437	0.9440	480
	Support Vector Machine (SVM)	0.9146	0.9193	0.9146	0.9151	480
	Random Forest (RF)	0.9000	0.9020	0.9000	0.9004	480
	Decision Tree (DT)	0.6354	0.7229	0.6354	0.6339	480
operation	RoBERTa	1.0000	1.0000	1.0000	1.0000	320
	Logistic Regression (LR)	1.0000	1.0000	1.0000	1.0000	320
	Support Vector Machine (SVM)	1.0000	1.0000	1.0000	1.0000	320
	Random Forest (RF)	0.9906	0.9906	0.9906	0.9906	320
	Decision Tree (DT)	0.9031	0.9132	0.9031	0.9025	320

## Robustness to Noisy Input

As discussed in Section 8.5.4, the data used to train and evaluate the models in this section does not fully capture the complexity of natural language—particularly with respect to non-standard forms and typographical errors. One way to mitigate this limitation is by improving both the training and testing datasets. Enhancing data quality in this way increases a model’s robustness to linguistic variation and improves the real-world relevance of performance evaluations.

Another complementary strategy is to select a classification model that is inherently more robust to noisy input. While LR, SVM, and RoBERTa achieve comparable performance on clean, synthetically generated test data, their behavior diverges under degraded conditions, as illustrated in Figure 8.5. In this setup, noise is simulated by randomly inserting characters—including letters, digits, and punctuation—into **greet** intent test utterances to mimic informal language and typographical errors. The number of inserted characters is calculated as a percentage of each utterance’s length, ensuring that the noise level scales proportionally with the input. Performance is evaluated and compared between models using recall, as it measures how well the model identifies the correct intent without being affected by predictions for other intents.

Under these conditions, RoBERTa’s recall for the **greet** intent remains relatively stable, whereas SVM’s performance degrades significantly—highlighting RoBERTa’s superior resilience to noisy input. This robustness likely stems from RoBERTa’s use of contextual embeddings learned through large-scale pretraining, which allow it to generalize more effectively to inputs that deviate from the training distribution. In contrast, SVM is more susceptible to noise and less capable of handling unfamiliar linguistic patterns. Although LR also exhibits a decline in performance under noisy conditions, the drop is noticeably smaller than that of SVM and only slightly greater than that of RoBERTa.

This makes it a solid classification choice, especially given its simpler and more lightweight architecture.

Recall of SVM, LR, and RoBERTa Relative to Noise in the greet Intent Test Dataset

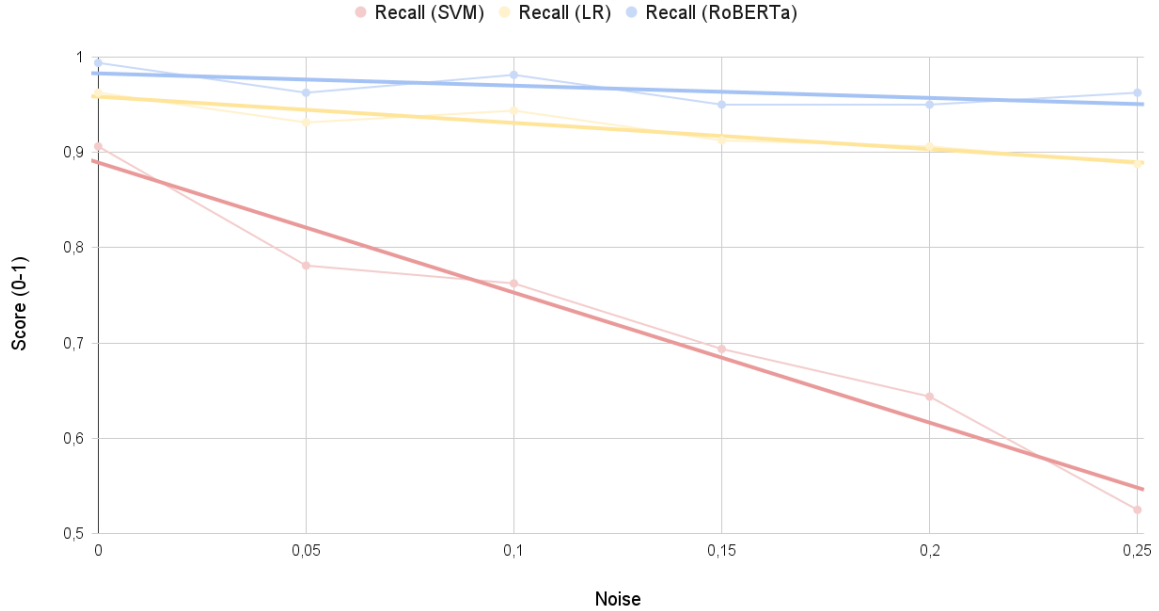


Figure 8.5. Recall values for the classification of the **greet** intent by LR, SVM, and RoBERTa on identical test sets infused with random character noise. Noise levels (0-0.25) indicate the proportion of inserted characters relative to the input length. Trend lines are included to improve readability.

From the results obtained in this performance evaluation section, performance in a real-world setting can be approximated by multiplying the DC accuracy with the IC accuracy. This reflects the fact that IC depends on correct DC in a deployed system, where a domain misclassification inherently leads to an incorrect intent classification.

It must be noted that the method used for inducing noise into the test datasets does not correspond to real-world non-standard forms or typographical errors, and therefore should not be considered fully representative of performance in practical settings. However, it effectively highlights the relative robustness of the best-performing models evaluated in this chapter, and more broadly, underscores the importance of model resilience when handling human—imperfect—language.

Full results—including confusion matrices—for DC and IC, as well as robustness-to-noise evaluations for IC, are provided in Appendices C.1, C.2, and C.6, respectively.

### 8.8.3 ER Results

Following the evaluation of domain and intent classification models, this section presents the results for ER, focusing on the extraction of **devicename** entities under various naming conditions.

## Evaluation Setup and Naming Variability

To simulate the variability of device naming conventions in real-world applications, two types of synthetic device names are used in this evaluation:

- **Random device names:** Arbitrary alphanumeric strings generated with varying lengths and combinations of valid characters (letters, digits, special characters such as +, -, and .), following the allowed character set defined by IoTOD EDM. These names contain no inherent structure or semantic patterns.
- **Realistic device names:** Synthetic names generated using real-world-inspired components such as common device roles (e.g., **router**, **switch**, **gateway**), still in-line with the allowed character set from IoT OD EDM.

## Experimental Configurations

Based on these naming schemes, six experimental configurations were defined to assess how well each model generalizes under varying training and testing conditions:

1. Train on randomly generated device names; test on randomly generated device names.
2. Train on randomly generated device names; test on realistic device names.
3. Train on realistic device names; test on randomly generated device names.
4. Train on realistic device names; test on realistic device names.
5. Train on a balanced mix of random and realistic device names (50/50); test on randomly generated device names.
6. Train on a balanced mix of random and realistic device names (50/50); test on realistic device names.

This setup enables a systematic evaluation of each model’s generalization capabilities. Realistic device names are representative of typical usage in real-world scenarios, making configurations 2, 4, and 6 the most indicative of practical model performance. In contrast, randomly generated names—while less common in real deployments—are still possible. Therefore, configurations 1, 3, and 5 are used to assess the model’s robustness to atypical device names.

## Evaluation Metrics

As in the evaluation for DC and IC, model performance is measured using the F1-score. Since the task involves a single entity type, and the resulting values for precision, recall, and accuracy are nearly identical across models, only the F1-score is reported here for clarity.

While sequence-level accuracy is also provided by MindMeld for ER, it is a stricter metric that reflects the correct labeling of an entire utterance. However, because this evaluation focuses solely on the extraction of a specific entity—namely, **devicename**—sequence accuracy is considered less relevant and is therefore omitted from the result interpretations.

Token-level F1-scores are calculated for each entity-related tag (e.g., `B|devicename`, `I|devicename`), and MindMeld reports a global support-weighted F1-score that reflects the distribution of those tags.

## Results and Analysis

As shown in Figure 8.6, MEMM and LSTM exhibit consistent performance across all configurations. CNN-LSTM and BERT yield the highest overall scores, though BERT shows a notable drop when trained on random names and evaluated on realistic ones. This is likely due to overfitting to irregular subword patterns introduced by WordPiece tokenization. When exposed only to random names during training, BERT may learn tokenization patterns that fail to generalize to more structured names during inference.

Overall, CNN-LSTM demonstrates the most consistent performance across all conditions, while BERT proves to be the most suitable model for both realistic deployment scenarios (configuration 4: realistic-to-realistic) and challenging edge cases (configuration 3: realistic-to-random).

Although MEMM and LSTM perform nearly as well as BERT, the latter offers superior robustness in handling unseen novel device names. This characteristic makes BERT particularly appealing for deployment in environments where the dataset cannot fully capture the evolving and unbounded nature of device identifiers, such as in the current context.



(a) Test performance on randomly generated device names. (b) Test performance on realistically structured device names.

Figure 8.6. F1-score comparison of ER models across six training and testing configurations involving synthetic device names. “*Random*” refers to arbitrarily generated strings with no semantic structure, while “*Realistic*” includes synthetically generated names that follow common naming conventions. Subfigure 8.6a shows performance when testing on random device names, and subfigure 8.6b shows performance when testing on realistic device names.

Full results for ER are provided in Appendix C.3.

## 8.9 Advantages and Limitations of Conventional NLU

### 8.9.1 Advantages

The NLU pipeline approach adopted in this chapter, using MindMeld and domain-specific ML models, offers several significant advantages:

- **Performance:** Overall, models such as LR and RoBERTa for IC, as well as CNN-LSTM and BERT for ER, demonstrated strong performance, validating the approach within the context of the defined IoT OD EDM use cases. This shows that, with curated and high-quality data preparation, along with the selection of robust models and carefully tuned parameters, the method explored in this chapter remains valid—even though a newer proof-of-concept approach is pursued in the next chapter for the CCV use cases.
- **Modular and Task-Specific Model Selection:** MindMeld separates DC, IC, and ER into distinct components. This modularity allows selecting and tailoring the most effective and resource-efficient model for each task individually, reducing overall computational cost.
- **Ease of Debugging and Optimization:** Due to its modular design, individual NLU components (DC, IC, ER) can be debugged, analyzed, and optimized independently. This separation simplifies troubleshooting and enhances maintainability of the overall pipeline.
- **Support for Role Classification (RC):** MindMeld provides an optional RC layer capable of distinguishing roles within similar entities—for instance, differentiating between `start_time` and `end_time` within the same message. While not utilized in this chapter, RC would have been useful for listing network activities within specified timeframes, which is a use case implemented in the next chapter.
- **Active Learning for Dataset Improvement:** MindMeld supports active learning [90], enabling iterative model enhancement through retraining on newly annotated, real-world data. This functionality effectively mitigates the limitations associated with synthetically generated training data by progressively incorporating authentic user interactions and more accurately representing non-standard linguistic expressions.

### 8.9.2 Limitations

Despite the outlined advantages, this approach presents several notable limitations:

- **Data Generation and Training Overhead:** The process of creating, annotating, and maintaining datasets, as well as configuring and training models, is resource-intensive and time-consuming. Expanding system capabilities by adding new intents or entities necessitates significant additional work, including data annotation and model retraining.
- **Challenges with Open-Class Entities:** Entities such as device names are inherently open-ended, making the use of gazetteers impractical. Consequently, entity recognition models rely entirely on learned patterns, limiting their generalizability to novel entity forms.

- **Complexity in Handling Relative Entity Formats:** Certain entities—especially date-type entities required in several use cases defined in Chapter 4—can be expressed in multiple ways, including absolute formats (e.g., *April 17, 2025*) and relative expressions (e.g., *yesterday, two weeks ago*). For conventional ML models, accurately interpreting such variations requires extensive annotated training data and the integration of external normalization modules, both of which are non-trivial to design and maintain.
- **Limited Support for Optional Entities:** While MindMeld can handle optional entities, it does not inherently differentiate between required and optional entities. Unbalanced training data (predominantly containing entity examples) may cause the model to mistakenly infer entities where none exist, leading to inaccuracies.
- **Sensitivity to Data Quality:** The system’s effectiveness heavily depends on the quality, comprehensiveness, and representativeness of the training data. Performance can degrade significantly due to noise, imbalance, or inadequate linguistic variation within datasets.

## 8.10 Chapter Summary and Practical Considerations

Throughout this chapter, we have seen that transformer-based models such as BERT or its variants such as RoBERTa are strong candidates for DC, IC, and ER. However, other models that are less computationally demanding—such as LR or MEMM—also demonstrate competitive performance in many cases.

Regardless of the chosen model, MindMeld provides the flexibility to mix different models across the layers of the NLU hierarchy. It also supports model serialization, allowing trained models to be easily exported and deployed within production environments such as Docker containers. Furthermore, MindMeld offers seamless integration of the NLU logic with the other components of the chatbot architecture.

When a Webex user message reaches the Chatbot Server component, it is forwarded to the Interpreter. The Interpreter processes the message by sequentially applying two models: one for DC and another for IC. These models classify the message first across high-level topics (called domains), and then within the appropriate domain to determine the specific intent. If the identified intent is eligible to contain entities, the message is then passed through a final ER model to extract relevant entities.

Figure 8.7 illustrates the flow between components, where the message remains unchanged as it passes through each model. Figure 8.8 presents a sample output from MindMeld’s NLU module after processing and parsing a user message.

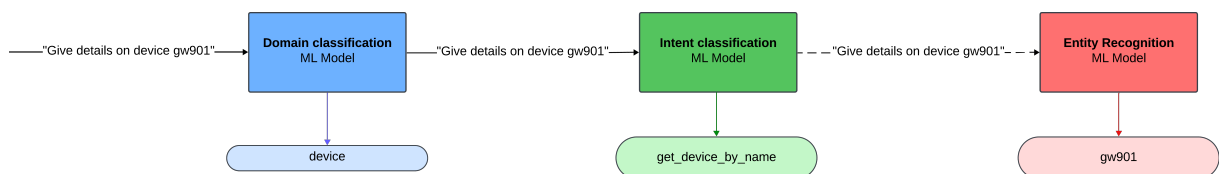


Figure 8.7. Message processing flow within the Interpreter component. The message is passed sequentially through the DC, IC, and, if applicable, ER models.

```

1 Request(
2     domain='device',
3     intent='get_device_by_name',
4     entities=(
5         immutables.Map({
6             'role': None,
7             'type': 'devicename',
8             'span': {'start': 30, 'end': 39},
9             'text': 'router123.',
10            'value': []
11        }),
12     )
13 )

```

Figure 8.8. MindMeld custom-tailored `Request` object produced by the NLU module after parsing a user message. It contains, among other fields, the results of DC (`device`), IC (`get_device_by_name`), and ER (`router123`), which are used by the Handler for generating a suitable response.

While the NLU pipeline approach discussed in this chapter adequately addresses the defined use cases and remains a valid strategy for building conversational AI-powered chatbots, it exhibits limited scalability and flexibility. Expanding the system to handle more complex or additional use cases would necessitate substantial adjustments, including modifications to the existing NLU pipeline, extensive data annotation, model retraining, and significant reengineering of components. These inherent limitations drive the exploration of a more adaptable and flexible approach leveraging LLMs, which is the focus of next Chapter.

The source code of the partially developed chatbot for IoT OD EDM is publicly available on GitHub<sup>4</sup> [91].

---

<sup>4</sup>Because integration with IoT OD EDM has been discontinued in favor of CCV, this repository is primarily intended for experimentation with MindMeld and does not aim to provide full integration with IoT OD EDM.

# Chapter 9

## Interpreter: LLM-Based NLU for Intent Classification and Entity Recognition

In this chapter, we present a second approach to handling NLU tasks—specifically, IC and ER. While Chapter 8 focused on a conventional ML pipeline trained or fine-tuned on structured datasets, this chapter explores a more flexible and dynamic method based on LLMs.

LLMs, a class of models within GenAI, are capable of understanding and generating human language, making them well-suited for tasks such as IC and ER. Popularized by transformer-based architectures such as the Generative Pre-Trained Transformer (GPT) family, these models produce structured, coherent outputs from representations learned across vast, unstructured corpora—demonstrating strong performance across a wide range of NLP tasks.

We begin by outlining why LLMs—particularly GPT models—are well-suited for NLU tasks, as discussed in Section 9.1. Section 9.2 then presents the motivations for adopting LLMs in the implementation of the **Interpreter** component. This is followed by an exploration of prompt engineering in Section 9.3, the core technique enabling effective use of LLMs, with a particular focus on handling the intents defined in Chapter 4.

In Section 9.4, we evaluate the performance of the new Interpreter on the CCV use cases defined in Chapter 4, and compare it with the first Interpreter’s performance on the IoT OD EDM use cases.

Sections 9.5 and 9.6 examine the strengths and limitations of this approach and propose directions for future improvement.

The chapter concludes, as in Chapter 8, with a practical summary in Section 9.7.

Within the context of integration with the core platform—CCV—the Interpreter’s role has shifted to support the use cases defined in Chapter 4 through NLU. This contrasts with the previous implementation in Chapter 8, which targeted integration with the IoT OD EDM platform.

While the current implementation builds on previously developed components—namely



the Chatbot Server, the Authenticator, and the integration between the initial Interpreter and the Handler (discussed in Chapter 10)—it moves away from using MindMeld as the NLP engine. Instead, it introduces a custom module that integrates LLM capabilities to fulfill the NLU functionality of the `Interpreter`.

## 9.1 Suitability of LLMs for NLU Tasks

Advancements in generative transformer models, notably the decoder-only GPT series, which represent a major class of generative transformer models, have provided significant complements to encoder-only models like BERT. While both models utilize a transformer-based architecture with self-attention mechanisms, their structures differ fundamentally. BERT employs an encoder-only design with bidirectional attention, processing entire input sequences simultaneously, thus excelling in discriminative tasks such as classification and contextual understanding. In contrast, GPT models adopt a decoder-only architecture, utilizing Causal Language Modeling (CLM), which follows an autoregressive approach—generating text sequentially in a unidirectional (left-to-right) manner. This approach naturally fits tasks involving coherent sequence generation.

Both BERT and GPT are examples of LLMs—neural networks with hundreds of millions to billions of parameters, trained on massive text corpora to understand and generate human-like language. However, while BERT is optimized for language understanding and commonly used in discriminative tasks, GPT is designed for generative tasks and typically features a larger architecture, making it more broadly applicable in natural language generation.

GPT models are pretrained extensively on diverse textual data using BPE tokenization, efficiently capturing linguistic patterns and contextual nuances. Compared to BERT, GPT is typically trained on a much larger corpus and consists of significantly more parameters, contributing to its broader language understanding and generative capabilities. Their key advantage for NLU tasks lies in their remarkable flexibility and adaptability to prompt-driven applications. By carefully engineering prompts, GPT can dynamically produce structured outputs, such as IC and ER. This capability significantly reduces the need for extensive task-specific fine-tuning, unlike conventional discriminative models (e.g., DT, LR, MEMM) or encoder-based transformers like BERT, which typically require additional task-oriented training layers.

Thus, the generative paradigm of GPT makes it particularly suitable for NLU tasks within chatbot applications, offering a powerful blend of versatility, simplicity, and effective contextual understanding [82, 92].

## 9.2 Motivation for Adopting LLMs

The transition from the conventional approach powered by MindMeld (as presented in Chapter 8) to a LLM-based solution is driven by two primary motivations: a shift in the project's scope and evolving technical considerations.

### 9.2.1 Shift in Project Scope

The project initially aimed to build a conversational AI-powered Webex chatbot integrated with IoT OD EDM, a platform for managing devices in an IoT network. In a subsequent phase, the focus shifted to a different platform: CCV, a security solution tailored for OT environments. While the core objective—developing a conversational AI chatbot—remains unchanged, this transition naturally brought with it an expanded interest in exploring how GenAI, and more specifically LLM, could be leveraged within this context.

This evolution is not only a response to changing platform requirements but also an opportunity to investigate modern AI capabilities. LLMs are central to the Assistant component’s NLG functionality (Chapter 11), and its potential is also examined for addressing NLU, originally implemented using conventional methods in Chapter 8. The shift toward generative techniques emerged as a logical progression, aligning with the expanded project scope and the increasing prominence of these technologies in contemporary conversational AI applications.

### 9.2.2 Technical Exploration

While the conventional ML approach described and implemented in Chapter 8 offers several advantages—including modularity, fine-tuning capabilities, independence from external services, and solid performance on unseen data thanks to robust transformer-based models such as BERT—it also presents notable limitations. These include the labor-intensive process of creating high-quality training data for each NLU task, difficulties in handling linguistic complexities such as non-standard language and relative formats, limited support for open-class and optional entities, and, more broadly, the time-consuming nature of implementing new use cases as the project scales.

It is within this context that the LLM approach is explored, with the aim of both mitigating these limitations and investigating a novel proof-of-concept technique for implementing NLU.

## 9.3 Prompt-Engineering Principles for IC and ER

In this section, we introduce prompts—a foundational concept in GenAI, enabling models across different modalities to adapt flexibly and efficiently to a wide range of tasks.

### 9.3.1 Role of Prompts

Prompt engineering arises from the architecture of GPT-type models, which are autoregressive language models trained to predict the next token given the preceding context. It involves the design of input prompts to steer the model’s behavior and shape its responses, aiming to improve performance on specific tasks. Prompts greatly influence the model’s performance, emphasizing the importance of carefully crafting them.

One of the key strengths of LLMs like GPT is their ability to generalize across a wide range of tasks without requiring task-specific fine-tuning such as the ML models we described in 8. This is largely due to their vast number of parameters and the diversity of their training data [92, 93].

Unlike conventional ways humans interact with computers—such as programming languages or structured query formats—LLM prompts are inherently *human-like*. They rely on natural language rather than strict syntax or formal rules. This paradigm arises from the fact that LLMs are trained predominantly on human-generated text, making them especially proficient at interpreting instructions phrased in natural language—including informal or non-standard expressions—thanks to the diversity of their training data.

A notable limitation in prompt engineering is the context window size—the maximum number of tokens (input plus output) the model can process at once. This means that all task instructions and generated responses must fit within this limit. In practice, an additional constraint is often imposed: the maximum number of output tokens the model is allowed to generate. This setting helps control the length of the response and ensures it stays within the context budget. As a result, prompt design must be space-efficient, balancing clarity, completeness, and performance without exceeding these limits. Albeit, research and development have significantly extended both of these limits. For instance, context windows have expanded from 16,385 tokens in models like GPT-3.5 to up to 128,000 tokens in advanced models such as GPT-4o. Similarly, the maximum number of output tokens has increased from 4,096 to 16,384 tokens, respectively [94]. As a result, these constraints are largely negligible—at least from a technical standpoint—for the NLU tasks addressed in this work where prompt sizes remain well within these limits.

### 9.3.2 Prompt Construction

In this section we construct the prompt designed to perform the IC and ER tasks of the Interpreter component. Unlike for the first approach of implementing the component, we do not perform DC before classifying the intent. This decision mainly stems from the fact that the number of distinct intents to recognize is relatively small and does not require an additional classification layer for domains.

The prompt has been abstractly divided into four parts:

1. **General Context Layer:** This part introduces the assistant’s role and its operational setting. It also defines the overall task: to determine the intent of a user message and extract any associated entities. By framing the assistant’s purpose early, we provide contextual grounding that helps the model stay aligned with the task.
2. **Intent and Entity Descriptions:** This section presents a structured list of all possible intents the LLM must recognize, along with the corresponding entities (if any) required for each. It serves as a kind of in-context schema, allowing the model to understand the expected label space.
3. **Specific Instructions:** We define constraints and logic that the model should apply when interpreting user input. For instance, we address how to handle optional entities, how to resolve relative dates, and how to interpret durations in the `subscribe_events` intent. These targeted instructions help reduce ambiguity and improve reliability on edge cases.
4. **Output Format:** Finally, we specify the expected output structure: a JSON object with two top-level keys, `intent` and `entities`. This format constraint ensures the model produces machine-readable outputs that can be parsed and processed

consistently by downstream components.

Figures 9.1-9.4 illustrate how each of these four parts appears, in order, in the actual prompt implementation.

```
You are a helpful assistant for a Cisco Webex bot. The bot receives user messages and uses
your output to determine the intent of the message and identify any necessary entities.
Your task is to categorize each user message into one of the predefined intents and extract
any required entities.
There can be typos in the messages, so always try to match an intent other than the 'unknown'
intent.
```

Figure 9.1. General Context Layer of the prompt.

```
The possible intents and their corresponding entities are:
- greet: Greeting.
- exit: Saying goodbye.
- help: Requesting help.
- list_presets: List presets with specific tags. Entities: tags (array)
- preset_details: Provide details about a preset. Entities: preset_label
- activate_preset: Activate a preset. Entities: preset_label
- deactivate_preset: Deactivate the active preset.
- list_tags: List tags.
- explain_tag: Explain a tag. Entities: tag_label
- list_baselines: List baselines.
- baseline_details: Provide details about a baseline. Entities: baseline_label
- list_devices_components: List devices and/or components from a particular country. Entities:
vendor_country
- device_component_details: Provide details about a device or a component. Entities:
device_label
- list_groups: List groups.
- group_details: Provide details about a group. Entities: group_label
- list_sensors: List sensors.
- list_activities: Show activities between dates and devices. Entities: start_date, end_date,
device_label_1, device_label_2
- list_vulnerabilities: Show vulnerabilities with a score above a specified score for a
specific device within a given date range. Entities: start_date, end_date, device_label, score
- explain_vulnerability: Explain a vulnerability. Entities: vulnerability_cve
- vulnerability_component: Identify and return vulnerabilities associated with a specific
component, exploit type, or attack vector. Entities: component_label
- similar_vulnerabilities: Identify if there are vulnerabilities that require the same
mitigations.
- vulnerability_anecdote: Tell something about a random vulnerability.
- subscribe_events: Subscribe to receive events at specified intervals in seconds. Entities:
interval_time
- unsubscribe_events: Unsubscribe to receiving events.
- disconnect: When disconnecting. Ending the session.
- unknown: Unknown or unclassified request.
```

Figure 9.2. Intent and Entity Descriptions.

```

Important:
- All entities are optional. Only include them if mentioned.
- Include dates only if explicitly mentioned, in the format YYYY-MM-DDTHH:MM:SSZ.
- The current date is $current_date. Use it for relative date calculations. Assume the current year if no year is specified.
- Ensure start_date is before end_date. Correct if necessary.
- vendor_country must be a real country.
- A device_label can be composed of an IP address.
- A vulnerability may be referred to as a CVE.
- If a vulnerability does not begin with "CVE", it is probably a component.

For the subscribe_events intent:
- Convert time durations mentioned by the user into seconds.
- Handle various time expressions explicitly:
  - "Year": Interpret as 365 days (31,536,000 seconds).
  - "Month": Assume 30 days if not specified otherwise.
  - "Month and a Half": Interpret as 1.5 months (e.g., 45 days, 3,888,000 seconds).
  - "Half a Month": Interpret as 15 days (1,296,000 seconds).
  - "Week": 7 days (604,800 seconds).
  - "Day": 24 hours (86,400 seconds).
  - "Hour": 3,600 seconds.
  - "Minute": 60 seconds.

```

Figure 9.3. Specific Instructions section, where `$current_date` is dynamically replaced with the actual current date at runtime.

```

Output JSON with:
- "intent": the appropriate intent
- "entities": needed info as key-value pairs, or empty if none

Provide your answer in pure JSON format without any markdown, code blocks, or additional commentary.

```

Figure 9.4. Output Format specification.

The *intent and entity descriptions* are derived from their definitions in Chapter 4, where the use cases and scope of the chatbot were introduced. The *output format* requirement, on the other hand, is driven by downstream components—particularly the Handler—which relies on a structured JSON format to determine and execute the appropriate logic based on the recognized intent and extracted entities.

In contrast, the *specific instructions* result from an iterative, hands-on engineering process. It reflects a set of practical refinements developed in response to inefficiencies and edge cases encountered during implementation, embodying a form of “brute-force” prompt engineering aimed at maximizing robustness and reliability.

## 9.4 Performance Evaluation

To assess the effectiveness of the proposed Interpreter approach, three distinct evaluation processes were conducted, each addressing a different objective.

### 9.4.1 Performance on CCV Use-Cases

The first evaluation measures the performance of both IC and ER using a fixed dataset consisting of 20 message utterances per intent, totaling 520 utterances. These utterances were partially created manually and then augmented synthetically using an LLM with a prompt, as illustrated in Figure 9.5. The IC and ER evaluations are conducted independently.

```
Provide me with 16 other phrases in the same vein:

List all presets, please. {'intent': 'list_presets', 'entities': {}}
Can you list presets containing the DHCP tag? {'intent': 'list_presets', 'entities': {'tags': ['dhcp']}}
Hey, what presets are available on my network? {'intent': 'list_presets', 'entities': {}}
Hey, what presets are available on my network that contain the ARP, write_var, and Windows tags? {'intent': 'list_presets', 'entities': {'tags': ['arp', 'write_var', 'windows']}}
```

Figure 9.5. Prompt used to generate 16 additional utterances for the `list_presets` intent test dataset, based on 4 manually created examples. Other datasets were generated using the same prompt, adapted to each specific intent. For evaluation purposes, each utterance is followed by a JSON representation of the expected classification.

For IC, the following metrics were computed as averages across all test datasets:

- **Average accuracy:** 0.9942
- **Average precision:** 0.9912
- **Average recall:** 0.9942
- **Average F1-score:** 0.9927

The ER evaluation, on the other hand, achieved 99.42% accuracy across all entities.

These results demonstrate the near-perfect performance of the proposed LLM-based approach for the Interpreter component of the chatbot, with only 3 misclassified intents out of 520.

While these results provide an estimate of the average performance of the final Interpreter implementation—within the limits of the test data—they do not offer a basis for comparison with the first version of the Interpreter from Chapter 8, as the use cases differ. Therefore, a second performance evaluation is conducted to enable a meaningful comparison.

### 9.4.2 Application to IoT Operations Dashboard Edge Device Manager Use Cases

The second evaluation applies the Interpreter to the use cases specific to IoT OD EDM defined in Section 8.2. This experiment compares the performance of the LLM-based approach against the conventional method. To that end, the prompt, shown in Figure 9.6, was adapted to align with the IoT OD EDM use cases, following a format similar to the one defined earlier. Both IC and ER are evaluated independently.

You are a helpful assistant for a Cisco Webex bot. The bot receives user messages and uses your output to determine the intent of the message and identify any necessary entities. Your task is to categorize each user message into one of the predefined intents and extract any required entities. You must always determine an intent. The possible intents and their corresponding entities are:

- greet: Greeting.
- exit: Saying goodbye.
- help: Requesting help.
- get\_all\_configuration\_groups: List configuration groups.
- get\_all\_alerts: List alerts.
- get\_all\_events: List events.
- get\_all\_devices: List devices.
- get\_all\_device\_locations: List locations associated with devices.
- get\_all\_device\_applications: List device software applications.
- get\_device\_by\_name: Provide details about a device. Entities: device\_label.

Output JSON with:

- "intent": the appropriate intent
- "entities": needed info as key-value pairs, or empty if none

Provide your answer in pure JSON format without any markdown, code blocks, or additional commentary.

Categorize the following input text:

Figure 9.6. Prompt used by the LLM-based Interpreter for IC and ER evaluation on IoT OD EDM use cases.

As illustrated in Table 9.1, which opposes RoBERTa, the best performing model from the previous chapter and GPT-4.1-mini-2025-04-14, a GPT model from OpenAI [95], the proposed approach demonstrates strong performance in classifying intents. Indeed, both approaches demonstrate similar results, the difference in performance not being representative as it can depend on factors independent from the model, such as the quality of the testing dataset.

Table 9.1. Performance Comparison of RoBERTa and GPT-4.1-mini Across Intent Classification Tasks

Domain	Model	Accuracy	Precision	Recall	F1-Score	Support
configuration	RoBERTa	1.0000	1.0000	1.0000	1.0000	160
	GPT-4.1-mini-2025-04-14	1.0000	1.0000	1.0000	1.0000	160
device	RoBERTa	0.9953	0.9953	0.9953	0.9953	640
	GPT-4.1-mini-2025-04-14	0.9875	0.9908	0.9875	0.9891	640
greeting	RoBERTa	0.9854	0.9856	0.9854	0.9854	480
	GPT-4.1-mini-2025-04-14	0.9979	0.9979	0.9979	0.9979	480
operation	RoBERTa	1.0000	1.0000	1.0000	1.0000	320
	GPT-4.1-mini-2025-04-14	1.0000	1.0000	1.0000	1.0000	320

Additionally, in terms of ER, the LLM-based Interpreter outperforms the conventional ML approach by achieving 100% accuracy. This improvement is attributed to the strong contextual understanding capabilities inherent in GPT models.



### 9.4.3 Noise Impact Comparison with RoBERTa

The final performance evaluation aims to assess the robustness of this approach to noisy input utterances.

As depicted in Figure 9.7, the GPT-based model, fed with the prompt from Figure 9.6, demonstrates the strongest robustness to noise, surpassing the already strong performances of RoBERTa.

Recall of SVM, LR, RoBERTa, and GPT-4.1-mini-2025-04-14 Relative to Noise in the greet Intent Test Dataset

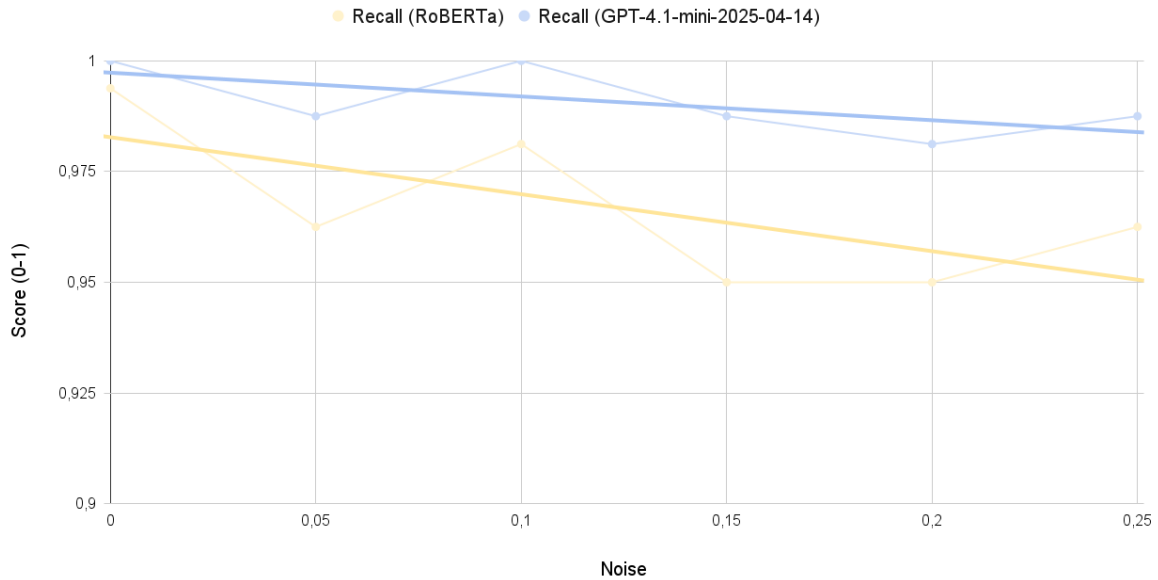


Figure 9.7. Recall values for the classification of the **greet** intent by RoBERTa and GPT-4.1-mini-2025-04-14 on identical test sets infused with random character noise. Noise levels (0-0.25) indicate the proportion of inserted characters relative to the input length. Trend lines are included to improve readability.

Full results for IC on CCV use cases and IoT OD EDM, as well as robustness-to-noise evaluations for IC, are provided in Appendices C.4, C.5, and C.6, respectively.

## 9.5 Advantages and Limitations of LLMs

The LLM approach has proven to be a strong choice for both IC and ER, consistently outperforming, on average, the method presented in Chapter 8. This section outlines the key advantages and limitations of the LLM-based Interpreter unrolled in this chapter.

### 9.5.1 Advantages

- **Performance:** As demonstrated by the evaluation experiments conducted in this chapter—both within the project scope and for the IoT OD EDM use case—the LLM approach delivers convincing performance across all test datasets. Even in



noisy environments, such as those involving non-standard language, the GPT models exhibit strong robustness.

- **High Flexibility, Adaptability, and Reduced Training Overhead:** Introducing new use cases—mainly in the form of new intents—requires only minimal modifications to the prompt. Typically, this involves adding a new intent and, if necessary, updating the instructions. Unlike conventional approaches, there is no need to collect new datasets, retrain the model, or perform extensive validation for every update. This makes the system highly adaptable and significantly accelerates development. As a result, the overall training overhead is greatly reduced, which is particularly beneficial in dynamic environments with evolving requirements.
- **Accurate Entity Recognition:** The results show near-perfect ER performance, despite the open-class nature of most entities. While BERT already achieved solid results, open-class entities introduced uncertainty and challenges in curating accurate training datasets. The LLM approach handles this more effectively.
- **Optional Fine-Tuning:** Although fine-tuning remains a possibility—further discussed in Section 9.6—the GPT-based model evaluated in this chapter already delivers strong out-of-the-box performance. This enables fast extension to new use cases and reduces dependence on high-quality, task-specific training data.
- **Built-in Handling of Relative Entity Formats:** A key limitation discussed in Chapter 8 was the handling of relative entity formats. In contrast, the LLM-based Interpreter, as detailed in Section 9.3.2, handles these cases through prompt instructions. Specifically, the prompt for CCV use cases includes detailed guidance on how to normalize date entities such as `yesterday`, `17 of April`, or `17/04/2025` into a consistent format (i.e., `YYYY-MM-DDTHH:MM:SSZ`). This ensures consistency in downstream components, such as the Handler.
- **Support for Optional Entities:** Just like relative entities, optional entities can be effectively managed by adapting the prompt. By explicitly describing the optionality of specific entities, the model can be instructed to handle them gracefully without needing structural changes to the implementation.

## 9.5.2 Limitations

Despite its many advantages, this method also presents several limitations that may justify choosing the original Interpreter approach instead.

- **Prompt Sensitivity:** LLMs are highly sensitive to the prompts they are given. While prompt engineering follows certain best practices, it remains a non-deterministic and task-specific process. There is no strict or standardized methodology for designing prompts, and crafting an effective one often involves iterative refinement. As new use cases emerge, the prompt must be continuously adapted to incorporate task-specific requirements and edge cases.
- **Non-Determinism:** Although prompts can instruct the GPT model to follow specific formats—such as returning output in a defined JSON structure—the model’s behavior is inherently probabilistic and not guaranteed. This non-determinism can result in variability due to factors such as hallucinations, where the model generates plausible but incorrect information, or prompt injection (both intentional

and unintentional), where inputs manipulate or override the prompt’s intended behavior. These issues emphasize the need for robust prompt design and system safeguards to ensure outputs remain consistent, secure, and user-transparent.

- **Computational Cost:** One of the main limitations of this approach is its significantly higher computational demand. The advanced capabilities of GPT models come at the cost of substantial processing power. For example, inference with GPT-3 can require approximately 350 TFLOPS, compared to just 680 GFLOPS for the base BERT model—making GPT-3 roughly 515 times more computationally intensive [96].

Note that, in this work, access to GPT models was implemented through OpenAI’s API, which introduced a financial cost not present in the first Interpreter approach. However, this cost is not strictly tied to the use of LLMs; it results from the choice to use a service-based deployment. A similar cost could have arisen if the first approach had used cloud-based models instead of running locally. Conversely, LLMs—particularly open-access variants—can also be deployed locally, thereby eliminating or significantly reducing service-related expenses. Benchmarks have shown that open-access models can perform competitively with proprietary models such as those from OpenAI on a wide range of tasks [97]. A dedicated performance evaluation would be required to formally assess whether such models could effectively replace OpenAI’s API in the context of this chatbot. Therefore, the financial cost is more a consequence of the deployment strategy than of the model type itself, and is thus not considered a limitation.

## 9.6 Directions for Improvement

Additionally to improving the already existing components of the Interpreter implemented in this chapter such as improving the prompt or choosing a better performing models, which evolve rapidly, this section presents potential design or functional directions for improvements. Note that this section groups a list of potential improvements and does not have the vocation to be exhaustive but to provide insight into the extensive possibilities GenAI offers.

### 9.6.1 Few-Shot Prompting and Fine-Tuning

The prompt designed in this chapter follows a *zero-shot* setting, meaning it does not include any examples or demonstrations. This approach is feasible due to the large-scale pretraining of models such as those based on GPT, which have been further fine-tuned to follow natural language instructions effectively [98]. While this prompt is capable of accurately classifying intents, recognizing entities, and consistently producing structured JSON output, a potential improvement would be to adopt a *few-shot* setting. Few-shot prompting enhances model performance by providing concrete examples that clarify task expectations, reduce ambiguity, and guide the model toward more consistent behavior.

For instance, adding an example such as the one illustrated in Figure 9.8 demonstrates how to extract both the correct intent and the associated entity. Including several such examples can help the model better generalize to new inputs, particularly in edge cases or with non-standard phrasing [99].

```
Input: "Tell me more about CVE-2025-1000"  
Output: {"intent": "explain_vulnerability", "entities": {"vulnerability_cve": "CVE-2025-1000"}}
```

Figure 9.8. Example that can be added to the existing prompt in a few-shot setting.

Further improvements can also be considered through model fine-tuning. Similar to other LLMs such as BERT, the GPT model used in this work could be fine-tuned with task-specific training data. While few-shot prompting is often effective, it also comes with limitations: it increases the token count of the prompt, which reduces scalability, extends the required context window, and may degrade model performance in accurately recognizing intents and extracting entities [100].

In this work, fine-tuning was not pursued, as the out-of-the-box performance of the GPT-based model was sufficient for the defined use cases. However, if future work involves more complex or numerous use cases and a drop in performance is observed, a promising direction would be to combine the zero- or few-shot prompt with a fine-tuned version of the LLM.

### 9.6.2 Hierarchical Separation of NLU Tasks

In Chapter 8, NLU was implemented using a hierarchical structure based on domain, intent, and entity recognition. Specifically, when the Interpreter receives a message, it first uses a DC-trained model to infer the domain of the message, representing a broad topical category. Within the inferred domain, a separate IC-trained model classifies the intent of the message. Finally, if the identified intent involves entities, an ER model is used to extract them. This modular design enhances modularity and scalability by allowing new use cases to be added independently and reduces the complexity of IC by narrowing the set of possible intents for each domain.

This structure can also be adapted to the current approach by employing a dedicated LLM for each of the DC, IC, and ER tasks. This separation not only preserves the benefits of modularity and scalability but also reduces the prompt size for each model, potentially improving performance and efficiency through a clearer division of responsibilities.

Additionally, beyond the advantages offered by this hierarchical separation, it is also possible to combine both Interpreter approaches. As demonstrated in the performance evaluation of the first Interpreter in Section 8.8.2, the RoBERTa-based model performs nearly perfectly on the DC task. This indicates that, at least for the IoT OD EDM use cases, employing a more computationally intensive model such as GPT for DC is not strictly necessary. Therefore, to optimize resource usage and reduce computational cost, a hybrid configuration could be considered: a lightweight model such as RoBERTa can be used for DC, while GPT-based models handle the more complex IC and ER tasks. Variations of this mixed approach can be implemented depending on specific requirements.

### 9.6.3 Input Sanitization

Before passing a user message to the LLM responsible for the NLU tasks in the Interpreter component, input sanitization can help improve both the model’s effectiveness and its resilience to prompt injection attacks<sup>1</sup>. This process may involve basic text cleansing, such as removing unwanted or potentially harmful characters, or more advanced techniques such as gibberish detection [101]. Another approach is to delegate the sanitization task to an LLM itself, by prompting it to translate the input into a cleaner version that retains the original meaning [102].

In this work, the latter approach—using an LLM for sanitization—was implemented experimentally. However, it was not retained in the final configuration. The reason is that certain open-class entities, such as device names, can be atypical and may be mistakenly classified as gibberish or noise, leading the sanitization model to remove or alter them, which distort the original message and negatively impact the performance of the Interpreter model. Therefore, if input sanitization is to be reintroduced, this method should be further refined to preserve the integrity of the message.

## 9.7 Chapter Summary and Practical Considerations

Throughout this chapter, we have shown that GenAI-based models—and more specifically, LLMs—can be effectively employed as the core engine for the Interpreter. Performance evaluations presented in Section 9.4 demonstrate that, even without fine-tuning, a model such as GPT-4.1-mini-2025-04-14 slightly outperforms the best-performing models from the first Interpreter approach—RoBERTa for IC and BERT for ER. The combination of strong performance, accelerated development—enabled by the ease of adding new use cases through prompt engineering—and future improvement potential led to the adoption of the LLM approach in the final implementation of the chatbot’s Interpreter.

Similarly to how MindMeld inherently provides seamless integration of NLU logic with the rest of the system, the new LLM-based **Interpreter** also integrates smoothly with the existing architecture. It enables easy selection of the most suitable OpenAI model, leveraging OpenAI’s API service, with the only constraint of having an API key.

When a Webex room message reaches the Chatbot Server component, it is forwarded to the Interpreter, as illustrated in Figure 9.9. The Interpreter processes the message by sending it, along with the prompt defined in Section 9.3.2, to the selected OpenAI model. The model’s response is then parsed as JSON and forwarded to the **Handler** component.

---

<sup>1</sup>While system security was not the primary focus of this thesis, it is worth noting that prompt injection attacks are inherently mitigated by the design of the pipeline. Responses from the Interpreter LLM are not shown directly to the end user; instead, they are passed to the Handler, which expects a strictly formatted JSON structure to process the request. Any deviation from the expected format—such as might occur from malicious prompt injection—will be rejected by the Handler, effectively discarding unexpected output.

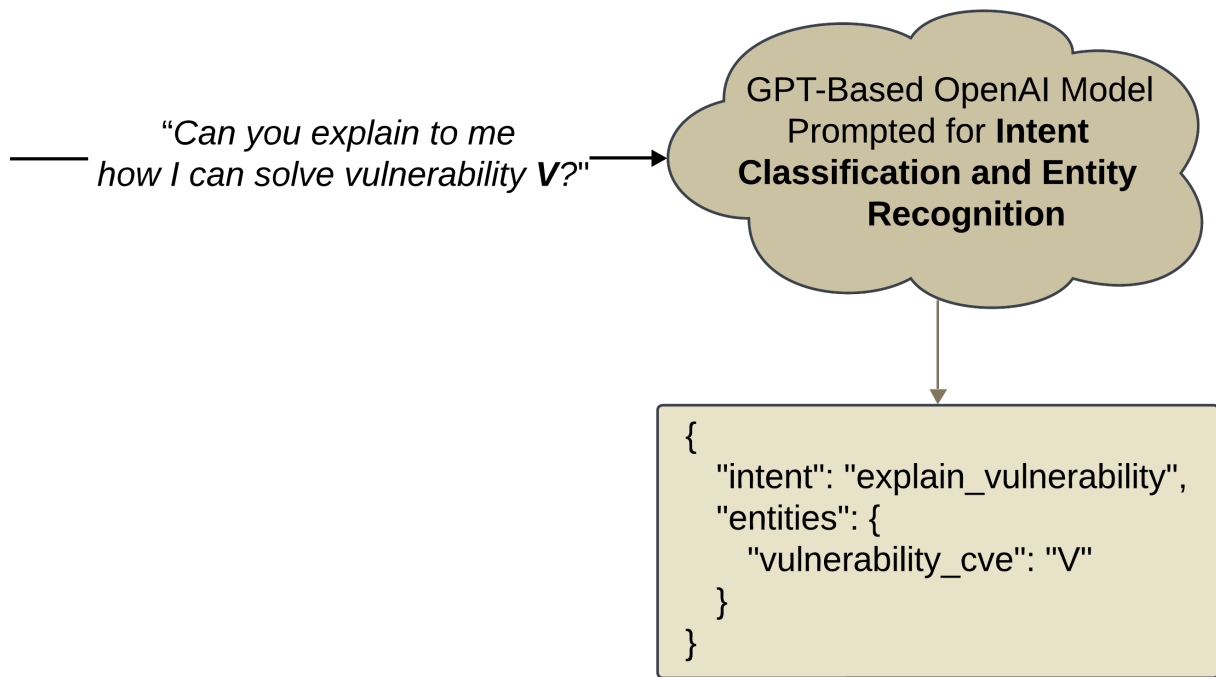


Figure 9.9. Message processing flow within the Interpreter component. The message is combined with a prompt and sent to the selected OpenAI GPT model.

# Chapter 10

## Handler

The Handler is the component responsible for determining which function to invoke based on the intent classified by the Interpreter.

In this chapter, we describe the overall structure of the Handler component. We explain the general design of a handler function, whose purpose is to manage a specific use case—namely, an intent—and introduce additional concepts that enhance the content and formatting of the chatbot’s responses.

### 10.1 Component Structure

Once the Interpreter has classified the intent of the received message and recognized the relevant entities, the Handler is invoked. The Handler serves as a central dispatcher, determining which action to execute based on the intent previously identified by the Interpreter (as described in the previous chapter).

It uses a decorator-based approach to register each handler function in a function registry—a dictionary that maps intent names (e.g., `list_presets` or `explain_vulnerability`) to their corresponding function.

When the Chatbot Server forwards a message to the Handler along with the classified intent and recognized entities, the Handler checks if the intent exists in the registry. If a match is found, it calls the corresponding function; if not, it returns a fallback response.

Intent functions are logically organized by domain, following the structure depicted in Figure 10.1.

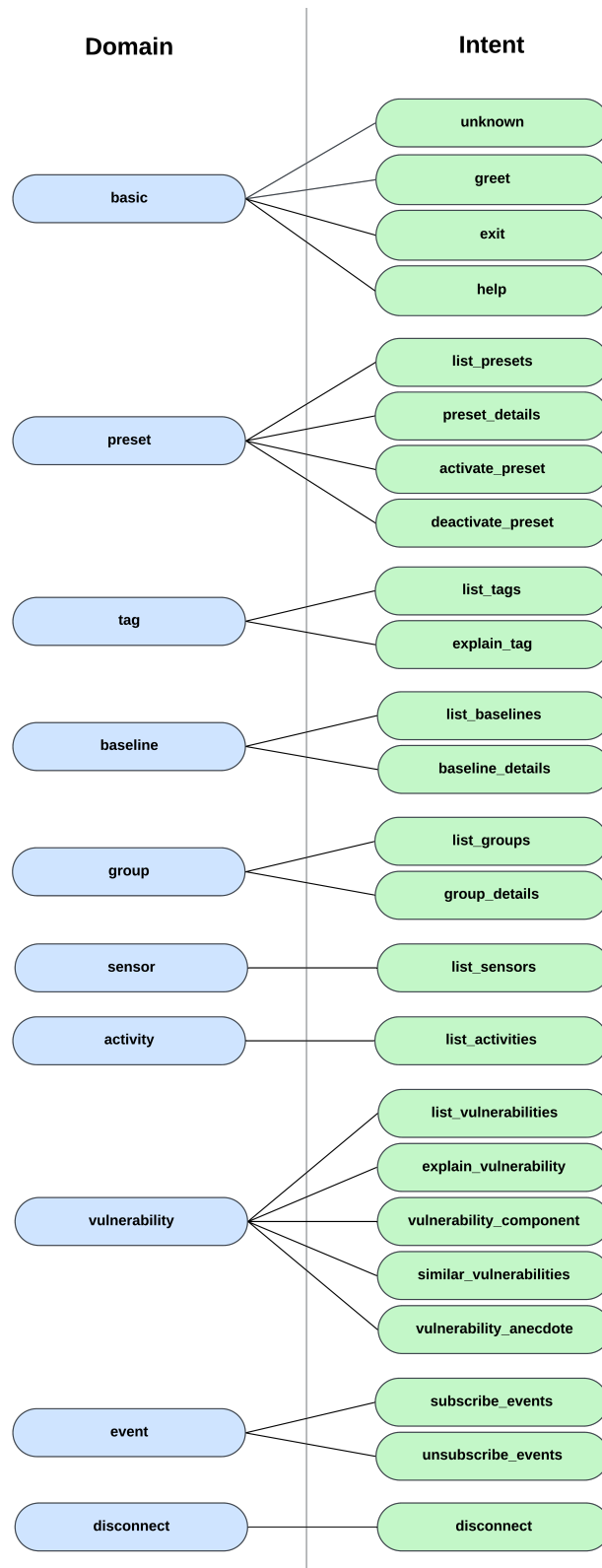


Figure 10.1. Logical organization of handler functions by domain.

## 10.2 Handler Function Structure

Handler functions can be categorized into three groups, corresponding to the three NLG techniques described in Section 3.3: template-based, rule-based, and LLM-based ap-

proaches.

### 10.2.1 Template-Based Approach

The template-based NLG approach consists in generating responses using predefined sentence patterns with placeholders that are filled in dynamically based on context. While this approach is very limited in flexibility and variability, it remains sufficient for simpler use cases represented by the **unknown**, **greet**, **exit**, and **help** intents from the **basic** domain. These use cases are not central to the chatbot’s scope and necessitate little logic.

All four intents follow a simple, predefined sentence structure. For example, when responding to a greeting, the chatbot randomly selects a message from a predefined list to deliver.

While this approach is sufficient for simple use cases, a more flexible rule-based strategy is necessary to handle most intents.

### 10.2.2 Rule-Based Approach

The rule-based approach consists in constructing responses through conditional logic and hand-crafted rules that adapt output based on specific input data or conditions. This method offers greater flexibility than template-based approaches but typically requires more implementation effort.

While the complexity of rule-based intent functions vary, most follow a consistent structure:

- When required by the use case, retrieve the recognized entities provided by the Interpreter to use in the logic of the subsequent steps.
- Retrieve the required data from the CCV Center’s API.
- When required by the use case, the recognized entity is mapped to actual CCV data in order to filter and adapt the response to the specific information requested—such as providing details for a given device or baseline.
- Define fallback logic to gracefully handle errors that may occur during the message process—such as unrecognized entities.
- Format the response to include the relevant content, dynamically inserting data retrieved from the API where appropriate.

While the rule-based approach is sufficient for many intent functions, some use cases require external GenAI assistance—particularly those involving the on-the-fly generation of new content that is not present in CCV. These use cases are addressed by the third NLG category: the LLM-based approach.

### 10.2.3 LLM-Based Approach

The LLM-based approach uses LLMs such as GPT models to generate a response given an input, instructed by a prompt. This method is used by the functions of the following intents:



- `list_devices_components`
- `explain_tag`
- `explain_vulnerability`
- `vulnerability_component`
- `similar_vulnerabilities`
- `vulnerability_anecdote`

While the handler function is responsible for invoking this functionality based on a given intent and using its output as a response, the actual content generation is delegated to a separate component—the Assistant. Due to its pivotal role in enabling the chatbot’s more advanced and novel functionalities—beyond what is possible through direct interaction with the CCV web UI—this approach will be discussed in detail in the next standalone chapter, which focuses on the LLM-based Assistant component.

## 10.3 Robustness and Extensibility Enhancements

To increase the chatbot’s tolerance to imperfect input and extend its feature set, several enhancements were implemented across the handler functions. These improvements focus on making entity handling more resilient and expanding the system’s capabilities in terms of time-aware queries, user event subscriptions, and flexible entity matching.

### 10.3.1 Time-Aware Entity Handling

Several use cases—such as retrieving activities or subscribing to events—require interpreting user-specified time constraints. These time-related entities are extracted by the Interpreter as part of the ER process and follow the ISO 8601 format (YYYY-MM-DDTHH:MM:SSZ). Internally, these date-time values are converted into Unix timestamps to enable accurate range filtering and temporal comparisons.

For example, the Handler function for the `list_activities` intent supports temporal filtering by processing optional `start_date` and `end_date` entities. These values are parsed and converted to timestamps that define the boundaries of the query. If no starting date is specified, the system defaults to retrieving all activities up to the current moment.

Similarly, the Handler function of the `subscribe_events` intent allows users to subscribe a room to periodic event notifications by specifying a time interval. This interval is validated and converted into a natural language expression (e.g., “*every 2 hours*”) to improve the clarity of the chatbot’s response.

These mechanisms enhance the chatbot’s ability to accurately process time-sensitive input and deliver user-friendly responses.

### 10.3.2 Persistent Event Subscriptions

In addition to responding to direct user queries, the chatbot is also capable of delivering periodic, event-driven updates to subscribed rooms. This functionality is supported by a

dedicated subscription mechanism that allows users to register and unregister their room for automatic notifications based on a predefined time interval.

Users interact with this mechanism through the `subscribe_events` intent, which validate the provided interval and add subscription metadata—including the next scheduled event time—in a dedicated database table. If the interval is missing or invalid, the system falls back to a default frequency.

For this use case, a background thread continuously monitors upcoming subscription triggers. When a subscription’s next scheduled event time is reached, the system retrieves relevant data from CCV—such as baselines and vulnerabilities—and sends an adaptive card update to the subscribed room. The card includes key metrics computed from the retrieved data, highlighting recent baseline changes or newly discovered vulnerabilities.

Subscriptions are stored persistently in a PostgreSQL table, allowing the mechanism to operate independently of the chatbot’s core request-response cycle.

### 10.3.3 Fuzzy Matching for Entities

To enhance robustness against misspellings and input variability, the chatbot employs fuzzy matching to resolve user-provided entities—such as device labels, tag labels, or vulnerability CVEs—that do not exactly match their actual label. This mechanism is particularly valuable in intents where precise identification is essential but exact string matches cannot be assumed.

Fuzzy matching relies on computing similarity scores between the user input and a set of candidate terms using Levenshtein distance—the minimum number of single-character edits required to transform one string into another [103]. The chatbot uses the `fuzzywuzzy` library [104] to perform these calculations. For example, a user input like “*devic1*” instead of “*device1*” may still yield correct results if the match score is sufficiently high.

To avoid unreliable matches, the system only accepts results that exceed a defined similarity threshold (e.g., 60%). Moreover, if multiple top-scoring candidates have identical scores, the match is discarded to prevent ambiguity.

This strategy ensures that minor input errors do not prevent the chatbot from handling a use case, while also safeguarding against incorrect or ambiguous matches.

### 10.3.4 Country-Aware Entity Handling

The `list_devices_components` intent supports filtering devices and components based on the country of origin of their vendors, whose names are specified in the device and component details.

To interpret country-related user inputs—whether specified as “*DE*”, “*DEU*”, or “*Germany*”—the chatbot applies a two-step normalization process. It first attempts to resolve the input using a custom dictionary that maps ISO 3166-1 alpha-2 and alpha-3 codes to full country names. This dictionary is derived from data provided by World-Data.info [105].

If no match is found, the system falls back to fuzzy matching against standardized country names using the `pycountry` library [106]. This ensures that a wide range of country input

formats can be correctly interpreted before proceeding with vendor-based filtering.

# Chapter 11

## Assistant

The final component of the chatbot system is the LLM-based Assistant—referred to simply as *Assistant* throughout this chapter. This component is invoked by the Handler whenever a use case requires a LLM-based approach to NLG. Such use cases are typically characterized by the need to generate textual responses that cannot be fully predefined or reliably constructed from static templates or structured data alone.

Tasks such as explaining a vulnerability or a tag, identifying similar vulnerabilities, or generating anecdotes could, in principle, be handled using rule-based systems and curated data sources. However, implementing such logic that way demands significant engineering effort and results in rigid, hard-to-maintain pipelines. By contrast, GenAI models—particularly LLMs—offer a more flexible and scalable solution. They simplify implementation while delivering natural, human-like responses. Although the use cases themselves are not inherently new, they are significantly enhanced by the expressive and adaptive nature of modern LLMs.

This component was designed to support that enhancement, helping address specific intents defined in the scope presented in Chapter 4. The handler functions that delegate to the Assistant are linked to the following intents:

- `list_devices_components`
- `explain_tag`
- `explain_vulnerability`
- `vulnerability_component`
- `similar_vulnerabilities`
- `vulnerability_anecdote`

In this chapter, we first describe the design and integration of the *Assistant* component within the Handler’s logic, highlighting its role in generating rich, natural language responses for intents that extend beyond rule-based or templated outputs. We then examine two functional extensions developed to enhance the Assistant’s capabilities: an intermediate database for caching responses, and a web search module for supplementing the LLM with up-to-date information. Building on this, we explore future directions

for improving the Assistant’s reliability and performance through Retrieval-Augmented Generation (RAG).

## 11.1 Component Design and Integration

The Assistant component is engaged by the Handler when an intent requires content generation that goes beyond predefined templates or rule-based responses—typically in cases where rich, natural language output is desirable.

The Assistant interfaces with an LLM (e.g., GPT) to generate context-aware responses. It constructs a prompt tailored to the specific interaction context—as illustrated in Figure 11.1—which is then processed by the underlying model to produce a dynamic reply.

During execution, the Handler gathers the relevant contextual information—such as a vulnerability entity requiring explanation—and delegates the task of response generation to the Assistant.

This design makes the Assistant a modular and reusable bridge for incorporating LLM capabilities into various intents.

```
You are a helpful assistant for a Cisco Webex bot. The bot receives user messages, and your task is to answer the following query concisely.
Your response should be well-formatted, provide relevant links when possible, and be easy to understand for novices.

Given the following tag and its description: Unite (Schneider Electric Unite is a protocol for managing and supervising Schneider Electric industrial devices); give a concise description and some simple use cases of the tag.
```

```
You are a helpful assistant for a Cisco Webex bot. The bot receives user messages, and your task is to answer the following query concisely.
Your response should be well-formatted, provide relevant links when possible, and be easy to understand for novices.

Tell a well-known anecdote or story about the vulnerability identified by CVE-2015-7871. Be concise and focus on any famous incidents or exploits related to this vulnerability or similar ones.
Format your response using markdown in an engaging way.
```

Figure 11.1. Example prompts constructed by the Handler for the Assistant, corresponding to the `explain_tag` and `vulnerability_anecdote` intents, respectively. The tag description in the first prompt has been intentionally abbreviated for illustrative purposes.

## 11.2 Functional Extensions

This section describes two extensions built on top of the Assistant’s LLM-based capabilities: an intermediate database used to cache its responses, and a web search integration that allows the model to retrieve up-to-date information in real-time when needed.

### 11.2.1 Intermediate Database for Response Caching

The `explain_vulnerability`, `explain_tag`, and `vulnerability_anecdote` intents share common characteristics: their responses do not depend on the room-specific authenticated CCV center, their responses are unlikely to change within a short timeframe, and they tend to produce relatively large outputs in terms of token count. This results in higher computational and financial costs when invoking the model. While this is an inherent consequence of using LLMs as opposed to smaller models, it is further amplified by relying on a paid OpenAI service in this particular implementation. Therefore, it is worth exploring ways to mitigate this cost.

Although several straightforward strategies exist—such as optimizing response length or selecting less resource-intensive models—the approach adopted in this thesis involves introducing an intermediate database to cache previously seen request responses. This database is shared across all chatbot rooms, allowing cached content to be reused globally.

When a message is classified as one of the above intents, the handler first checks whether an explanation for the given vulnerability—identified by its CVE identification value—or the specified tag is already present in the database, before invoking the Assistant to resolve the use case. If a cached explanation is found, it is used as the response to the room. Otherwise, the Assistant is used, and the resulting response for the given entity is subsequently cached in the database. Each entry is assigned a configurable expiration time to limit retention time and prevent information from becoming stale.

While caching provides notable performance and cost benefits—particularly in the context of frequently accessed data—it also introduces challenges, especially in maintaining data accuracy and freshness. This highlights the importance of carefully balancing the trade-off between expiration time (i.e., data freshness) and cost efficiency.

### 11.2.2 LLM Temporal Limitations Addressed through Web Search

#### LLM Temporal Limitations

An important consideration when using LLMs is the limitation of the information they implicitly capture from their training data. Since LLMs generate responses based on statistical patterns rather than verified facts, their outputs are not guaranteed to reflect ground truth. As a result, these models can produce inaccurate or fabricated information—a phenomenon known as hallucination. This issue becomes more likely when the model encounters inputs involving information that was not well represented or sufficiently learned during training [107].

While the inaccuracy of generated content must be acknowledged as a general limitation of GenAI in resolving the use cases addressed in this work, it becomes an even greater concern when dealing with information that the model definitively could not have encountered during training—specifically, data that appeared after the model’s training cutoff date, such as newly disclosed vulnerabilities. For example, GPT-4.1-mini-2025-04-14, which was used in the implementation of the LLM-based Interpreter, is unable to accurately fulfill the `explain_vulnerability` intent if the specified vulnerability was discovered after April 14, 2025, which marks the end of the model’s training period.

## Dynamic Web Search with LangChain

To overcome the limitations of the fixed training data, the Assistant is extended with dynamic web search capabilities. This allows it to retrieve up-to-date information in real time when needed, extending its knowledge beyond the static scope of the underlying LLM and enabling accurate responses on, for example, newly discovered vulnerabilities.

This functionality is implemented using the LangChain framework, which facilitates integration of LLMs with external tools like web search APIs [108]. In this implementation, the Tavily Search API is used [109]. When a search is triggered, real-time queries to Tavily retrieve relevant results, which are then incorporated into the prompt provided to the Assistant, which it uses to enhance its responses, ensuring they are grounded in the most current information available.

## 11.3 Directions for Improvement Using Retrieval-Augmented Generation (RAG)

Out-of-the-box LLMs are limited to the data they were trained on. To address this, web search capabilities were integrated, as described in Section 11.2.2. While effective for retrieving dynamic content—such as up-to-date CVEs—this approach introduces variability, latency, and reliability issues. As an alternative, this section explores the use of RAG, which offers a more structured and extensible method for incorporating external knowledge.

RAG enhances the system by retrieving relevant information from a curated knowledge base and combining it with the user’s message to produce grounded, context-aware responses. As illustrated in Figure 11.2, the RAG process proceeds as follows:

1. **Indexing:** External sources—such as CVE entries (e.g., from the National Vulnerability Database [110]), CCV documentation, vendor advisories, and device specification sheets—are chunked, embedded, and stored in a vector database for similarity-based retrieval.
2. **Retrieval:** At query time, the user’s question is embedded and matched against the database. This may retrieve CVE-related content.
3. **Prompt Construction:** The retrieved content is combined with the original query to form a structured prompt, which serves as input to the language model.
4. **Generation:** The LLM uses this context-rich prompt to generate a response grounded in the retrieved documents—reducing hallucinations and improving factual accuracy.

By combining precise retrieval with controlled generation, RAG offers a robust and scalable alternative to the less predictable web-based approaches [111].

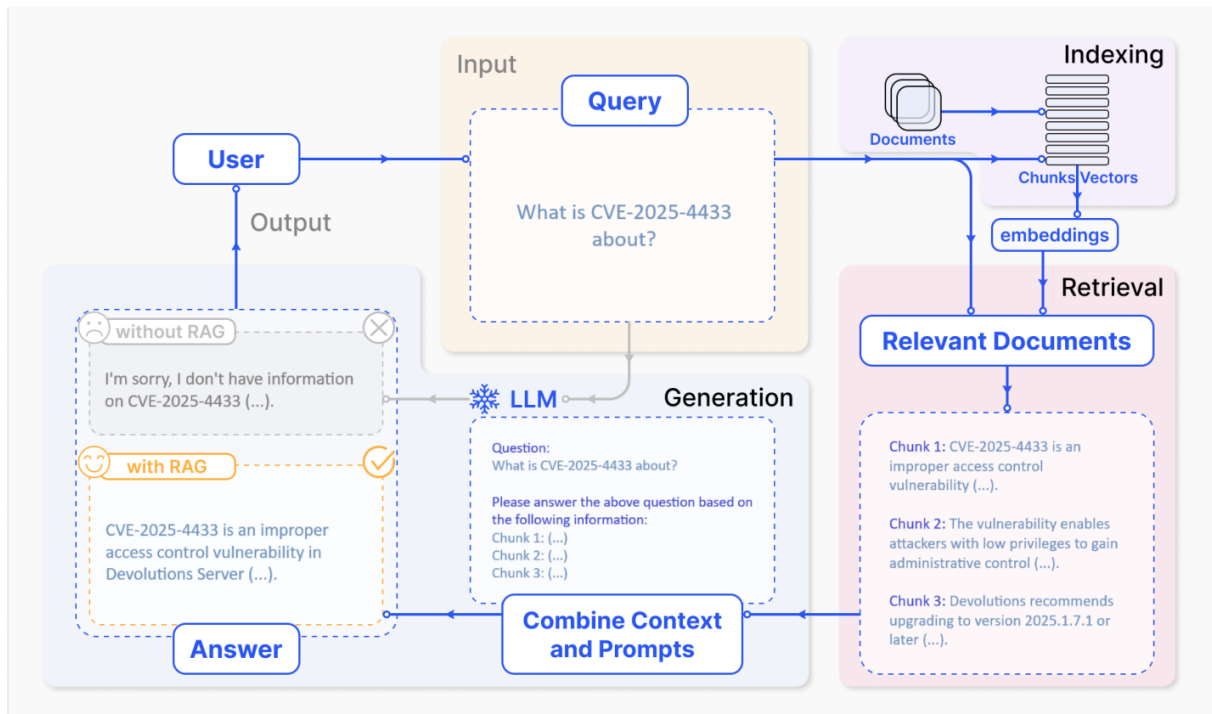


Figure 11.2. Overview of the RAG pipeline used to enrich LLM responses with external knowledge. This figure is adapted from Figure 2 of the paper *Retrieval-Augmented Generation for Large Language Models: A Survey* [111], and contextualized for the chatbot’s vulnerability explanation use case. The CVE-2025-4433-specific information was retrieved from the official advisory [112].



# Chapter 12

## Final Use Case Demonstrations: Webex User Viewpoint

This chapter presents visual examples of the use case resolutions defined in Section 4, demonstrating the system’s capabilities through screenshots captured from the Webex user’s perspective. These snapshots illustrate how the chatbot responds to various classified intents across different domains, as outlined in Figure 10.1.

The focus is on showcasing the user-facing interactions without delving into internal system logic. For clarity and conciseness, some screenshots have been cropped while preserving the essential content of each exchange.

The LLMs used for the *Interpreter* and the *Assistant* are, respectively, GPT-4.1-mini-2025-04-01 and GPT-4.1-2025-04-14.

### 12.1 Basic Use Cases

Figure 12.1 shows example interaction snapshots between a Webex user and the chatbot. These correspond to the **greet**, **exit**, and **help** intents. When the chatbot classifies a message as an **unknown** intent, it responds with a message similar to that used for the **help** intent.

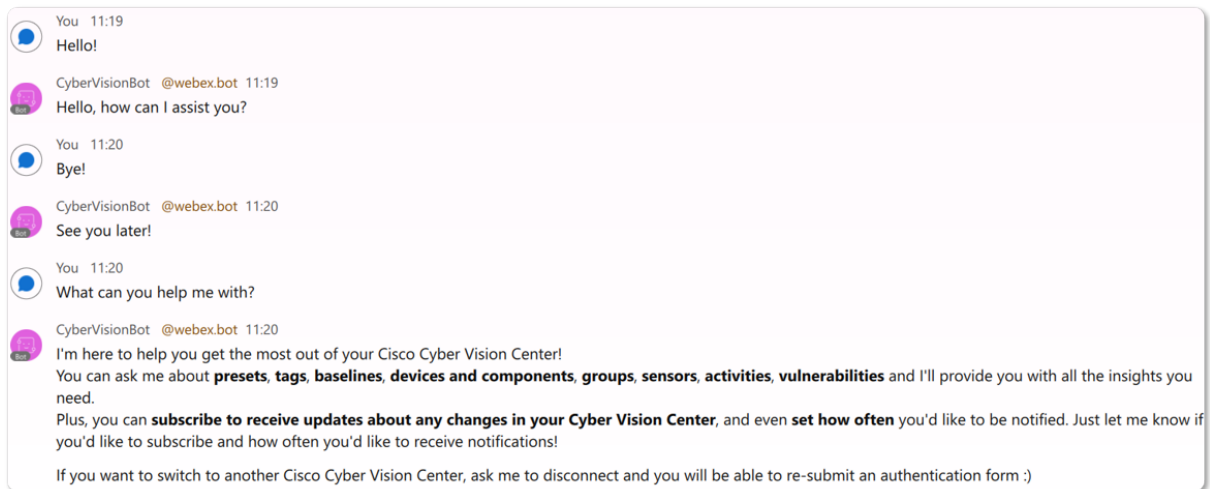


Figure 12.1. Illustrative chatbot interactions for the **greet**, **exit**, and **help** intents.

## 12.2 Use Cases Related to Presets

Figures 12.2-12.4 illustrate how the chatbot handles various preset-related intents. These include listing available presets, retrieving preset details, and activating or deactivating presets. For the **list\_preset** intent, users may specify one or more tags to filter the presets. If no tags are provided, all presets from the CCV center are returned.

When a preset is activated, its ID is stored in the corresponding room database entry, and subsequent chatbot responses for other use cases are interpreted within the context of that preset—much like how, in the CCV web UI, selecting a preset applies a predefined set of filters that scope the displayed data.

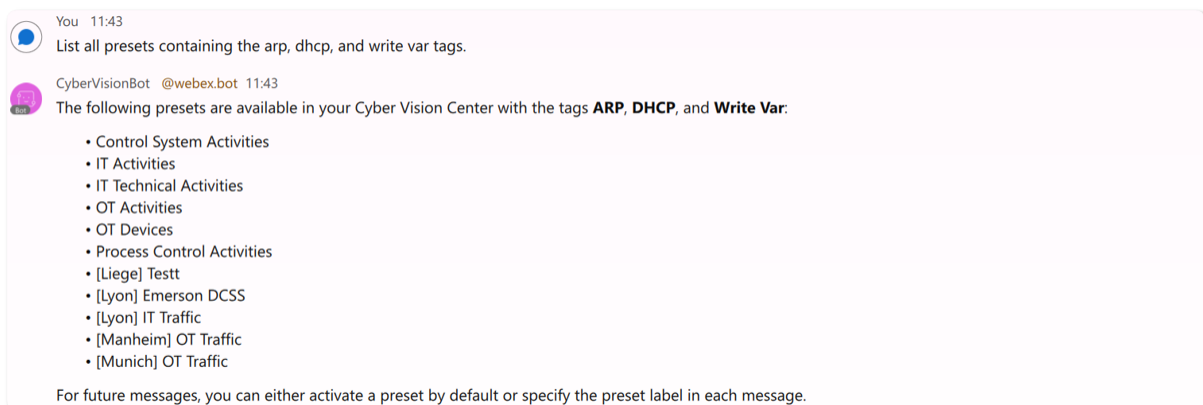


Figure 12.2. Illustrative chatbot interaction for the **list\_preset** intent.

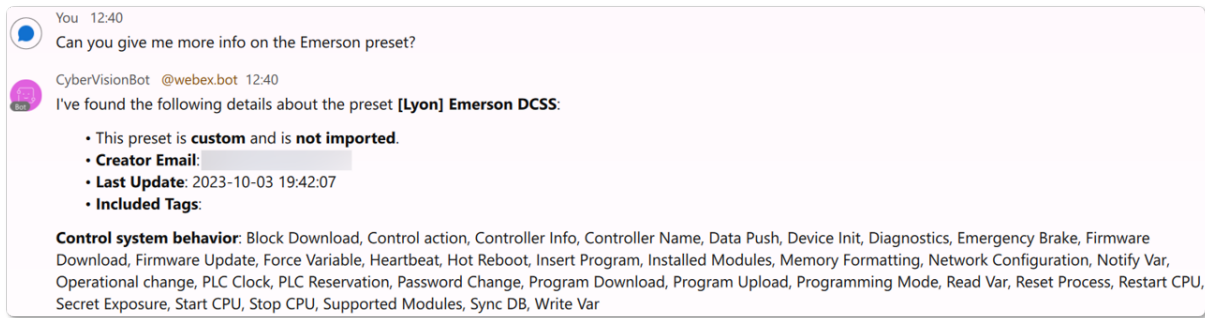


Figure 12.3. Illustrative chatbot interaction for the `preset_details` intent.

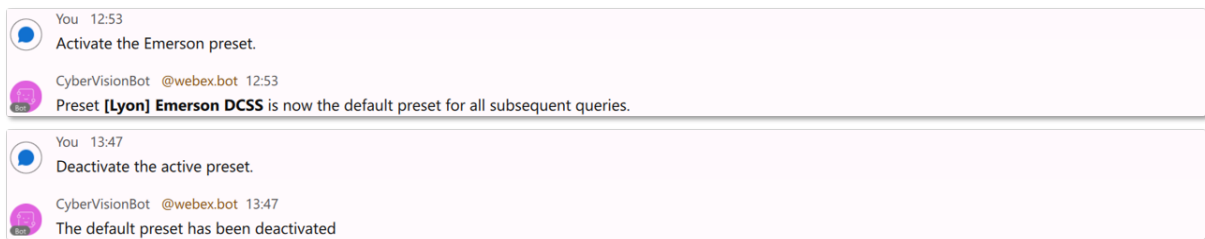


Figure 12.4. Illustrative chatbot interactions for the `activate_preset` and `deactivate_preset` intents.

## 12.3 Use Cases Related to Tags

Figures 12.5 and 12.6 show chatbot interactions related to tag-based functionality. The `list_tags` intent allows users to retrieve all relevant tags associated with an active preset. The `explain_tag` intent enables users to get descriptions or explanations for specific tags, assisting in better understanding and selection.

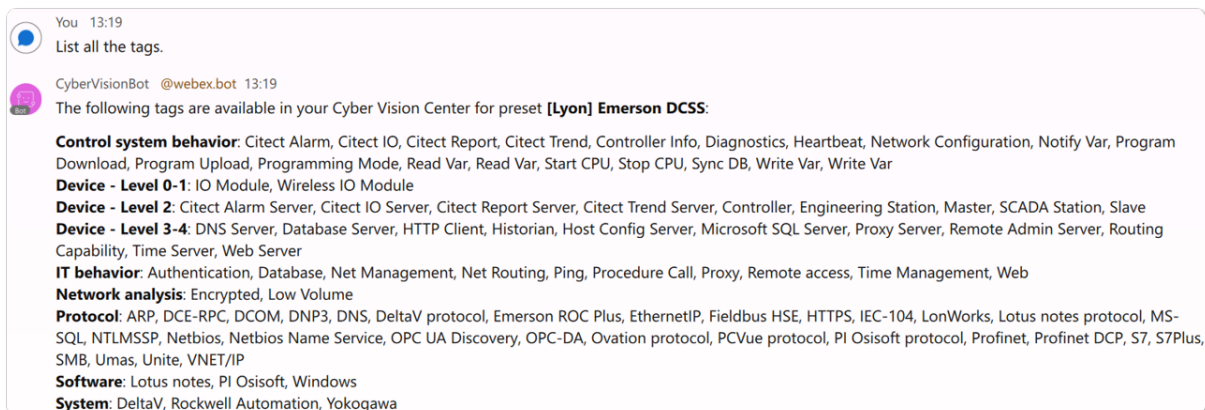


Figure 12.5. Illustrative chatbot interaction for the `list_tags` intent.

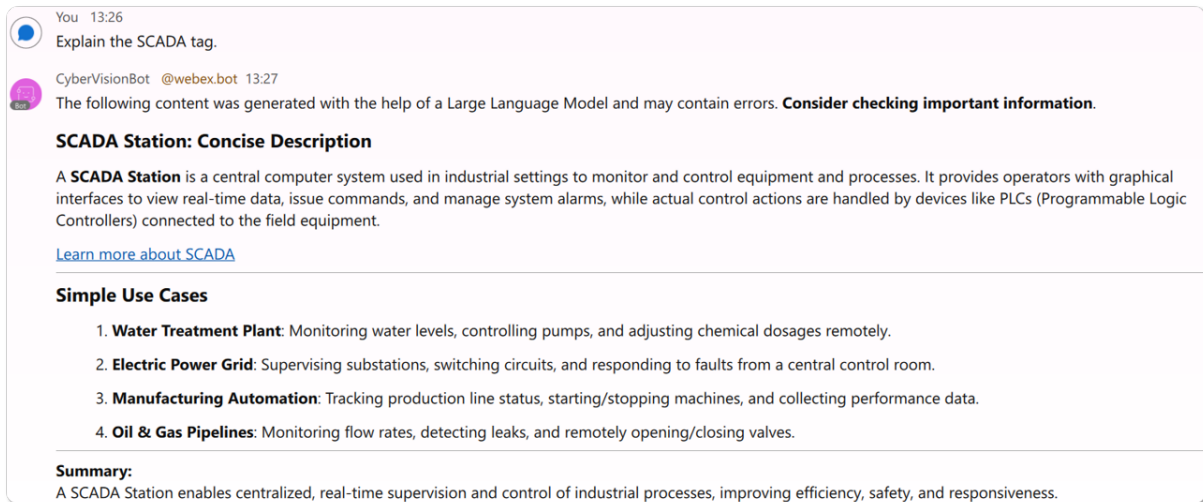


Figure 12.6. Illustrative chatbot interaction for the `explain_tag` intent.

## 12.4 Use Cases Related to Baselines

Figures 12.7 and 12.8 demonstrate how the chatbot supports baseline-related functionality. The `list_baselines` intent retrieves a list of available baselines, while the `baseline_details` intent provides additional information about a selected baseline.

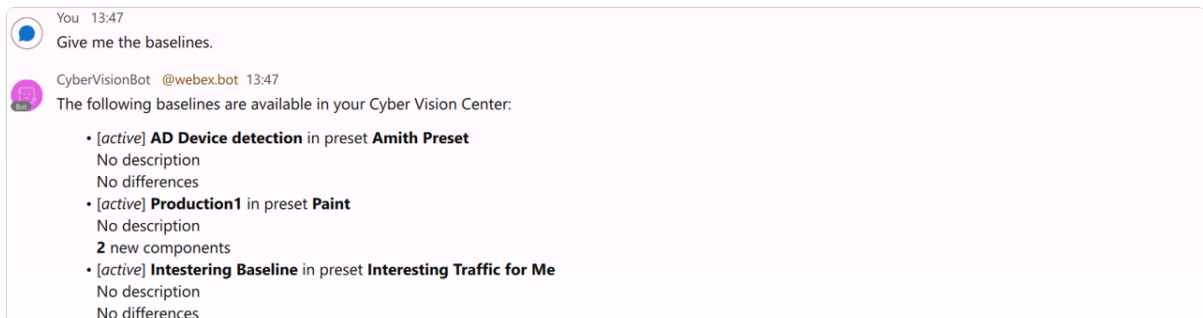


Figure 12.7. Illustrative chatbot interaction for the `list_baselines` intent.



Figure 12.8. Illustrative chatbot interaction for the `baseline_details` intent.

## 12.5 Use Cases Related to Groups

Figures 12.9 and 12.10 illustrate the chatbot’s functionality for managing groups. The `list_groups` intent allows users to retrieve an overview of available groups, while the `group_details` intent provides detailed information about a selected group.

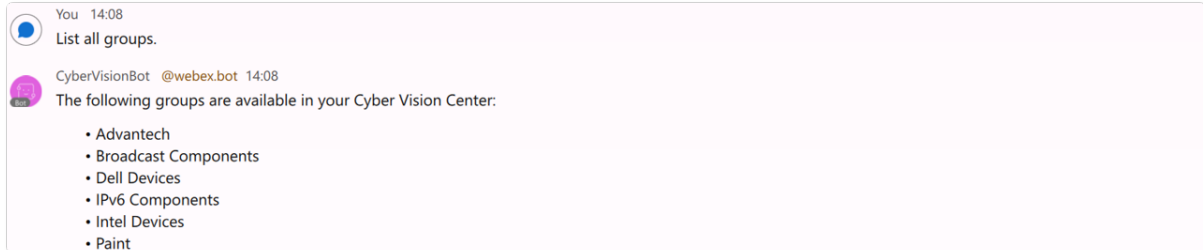


Figure 12.9. Illustrative chatbot interaction for the `list_groups` intent.

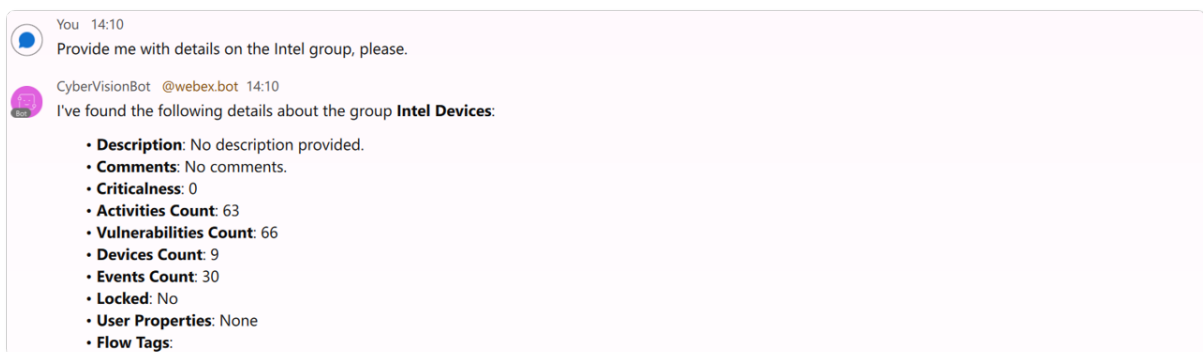


Figure 12.10. Illustrative chatbot interaction for the `group_details` intent.

## 12.6 Use Cases Related to Sensors

Figure 12.11 shows an example of how the chatbot handles sensor-related messages. Specifically, the `list_sensors` intent allows users to retrieve a list of sensors currently available in the system.

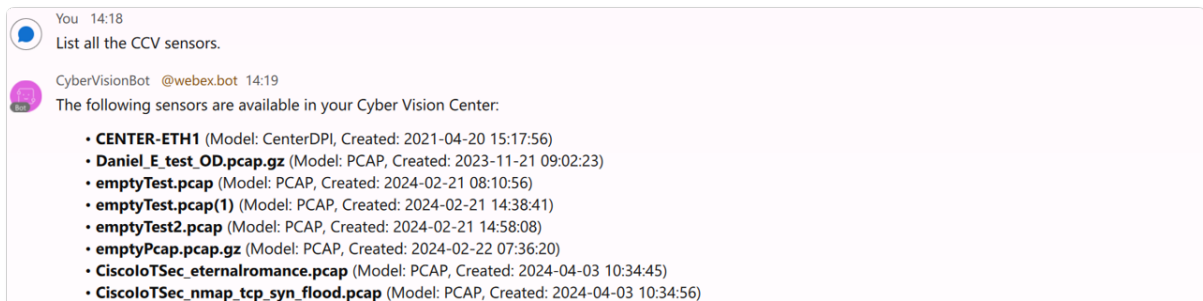


Figure 12.11. Illustrative chatbot interaction for the `list_sensors` intent.

## 12.7 Use Cases Related to Devices

Figure 12.12 illustrates the chatbot’s support for device-related messages. The `list_devices_components` intent enables users to retrieve available devices and components. A vendor origin may be specified to filter the results; if omitted, all devices and components from the CCV center are returned.

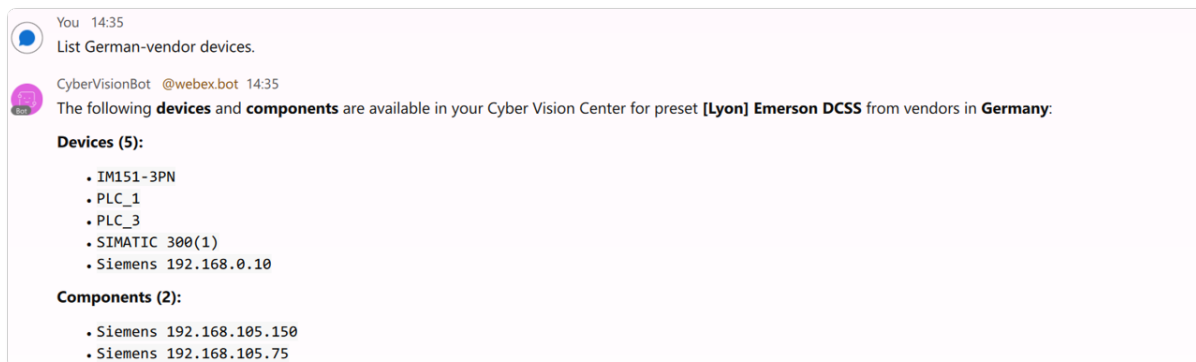


Figure 12.12. Illustrative chatbot interaction for the `list_devices_components` intent.

## 12.8 Use Cases Related to Activities

Figure 12.13 presents an example of how the chatbot handles activity-related messages. The `list_activities` intent allows users to retrieve a list of activities, optionally filtered by up to two devices or components and a timeframe. If no filters are provided, all activities from the CCV center are returned.

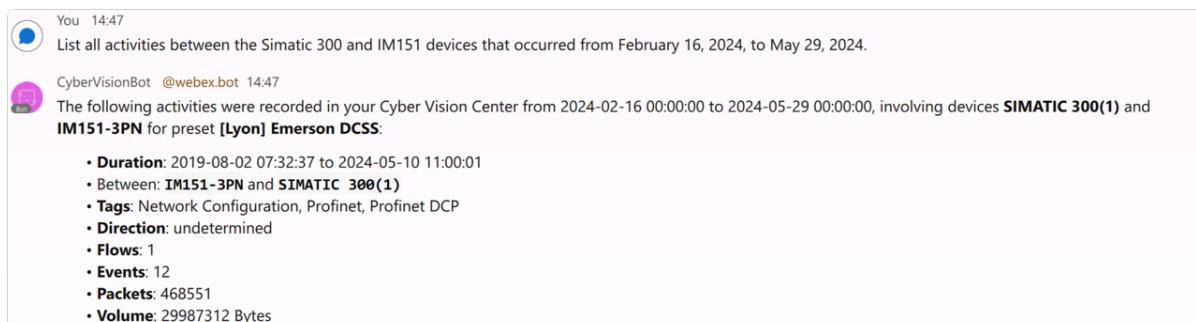


Figure 12.13. Illustrative chatbot interaction for the `list_activities` intent.

## 12.9 Use Cases Related to Vulnerabilities

Figures 12.14-12.18 illustrate how the chatbot handles vulnerability-related intents. The `list_vulnerabilities` intent enables users to retrieve vulnerabilities, optionally filtered by a specific device or component and a CVSS score. The `explain_vulnerability` intent provides a plain-language description of a given vulnerability. The `vulnerability_component` intent allows users to find vulnerabilities associated with a specific component, while the `similar_vulnerabilities` intent returns related or similar issues. Lastly, the `vulnerability_anecdote` intent presents contextual anecdotes or historical notes associated with a particular vulnerability.

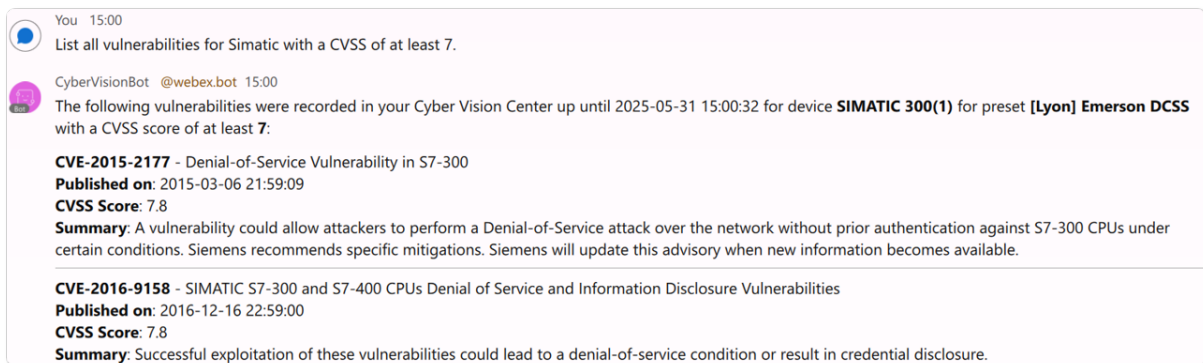


Figure 12.14. Illustrative chatbot interaction for the `list_vulnerabilities` intent.

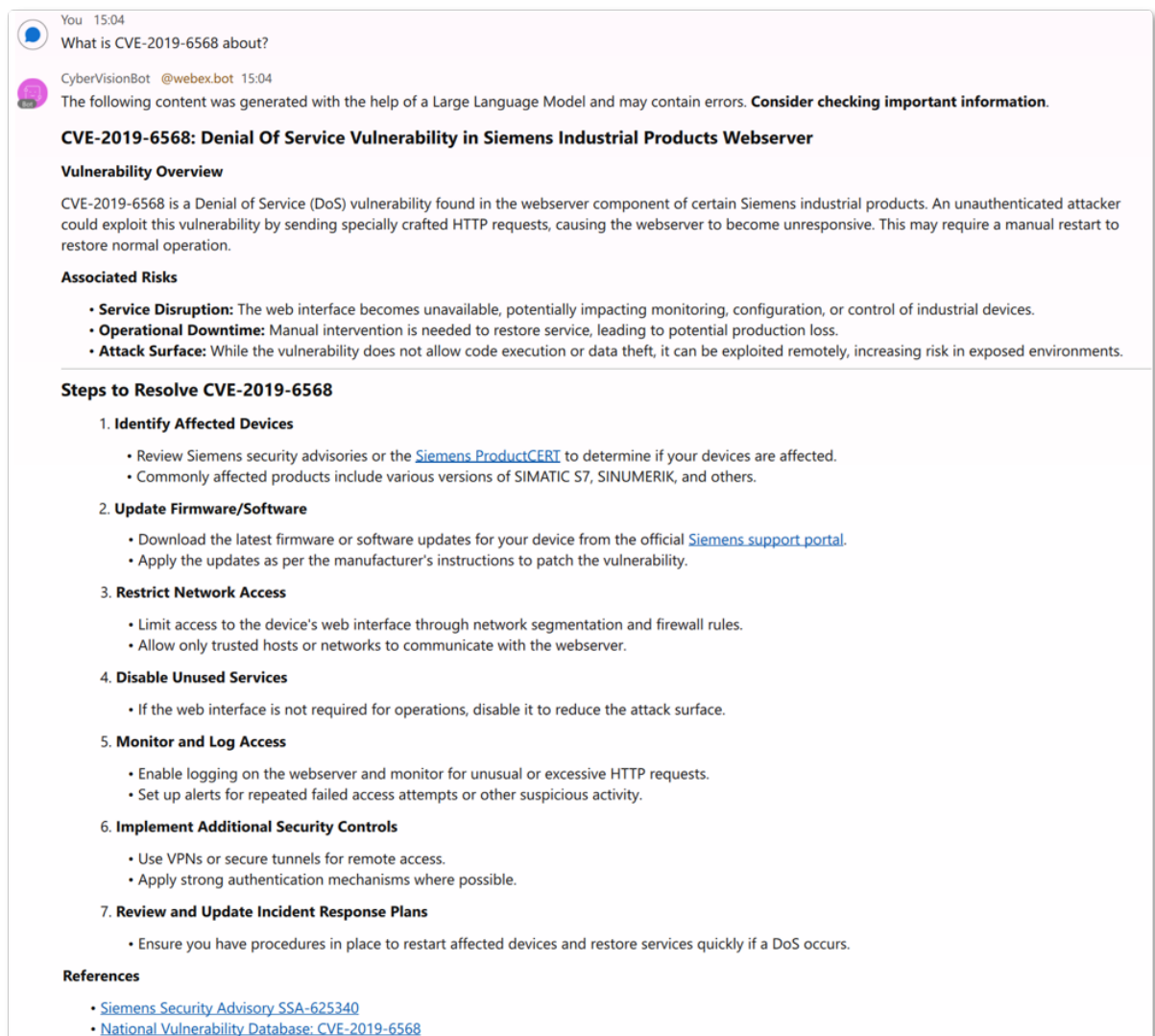


Figure 12.15. Illustrative chatbot interaction for the `explain_vulnerability` intent.



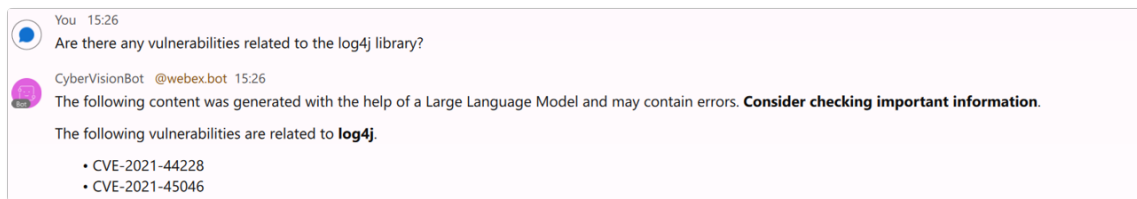


Figure 12.16. Illustrative chatbot interaction for the `vulnerability_component` intent.

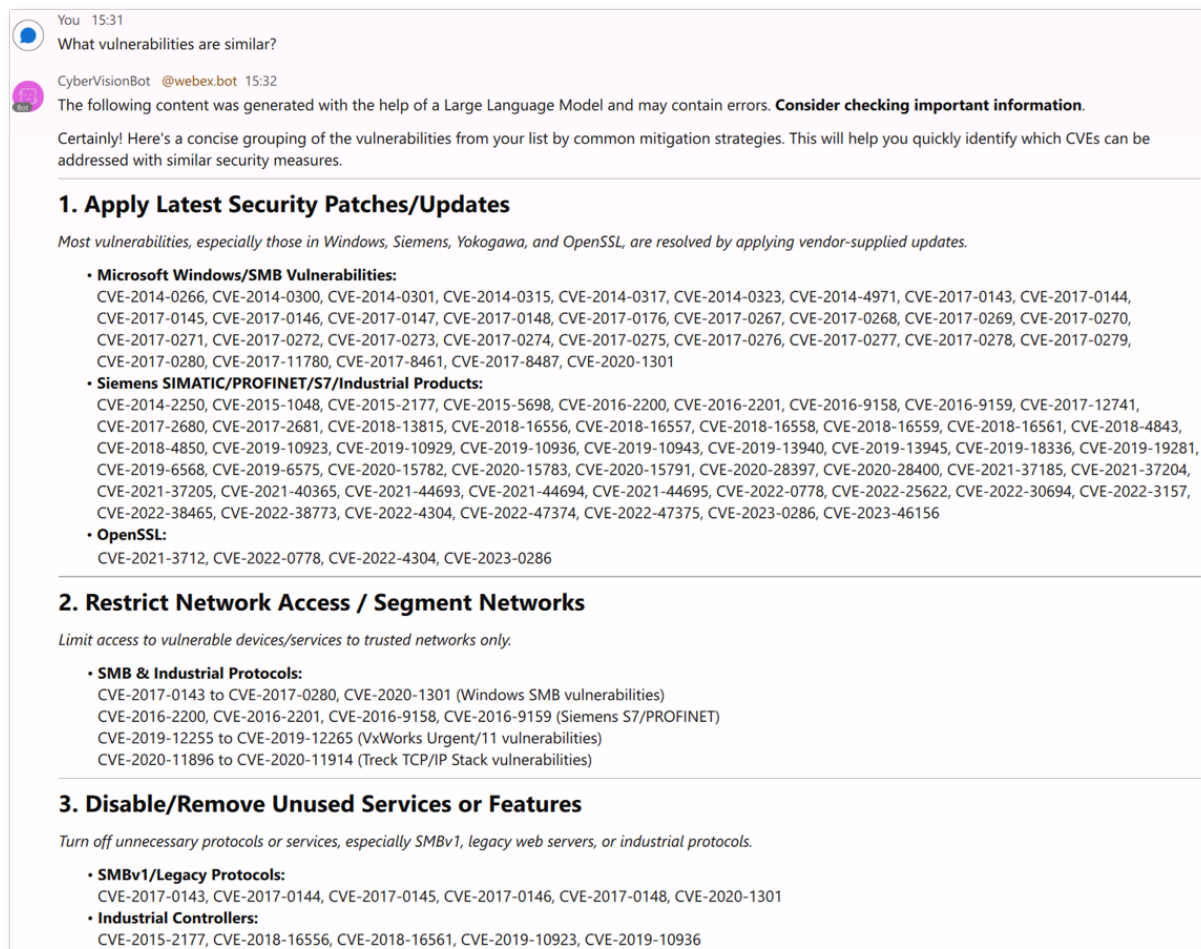


Figure 12.17. Illustrative chatbot interaction for the `similar_vulnerabilities` intent.



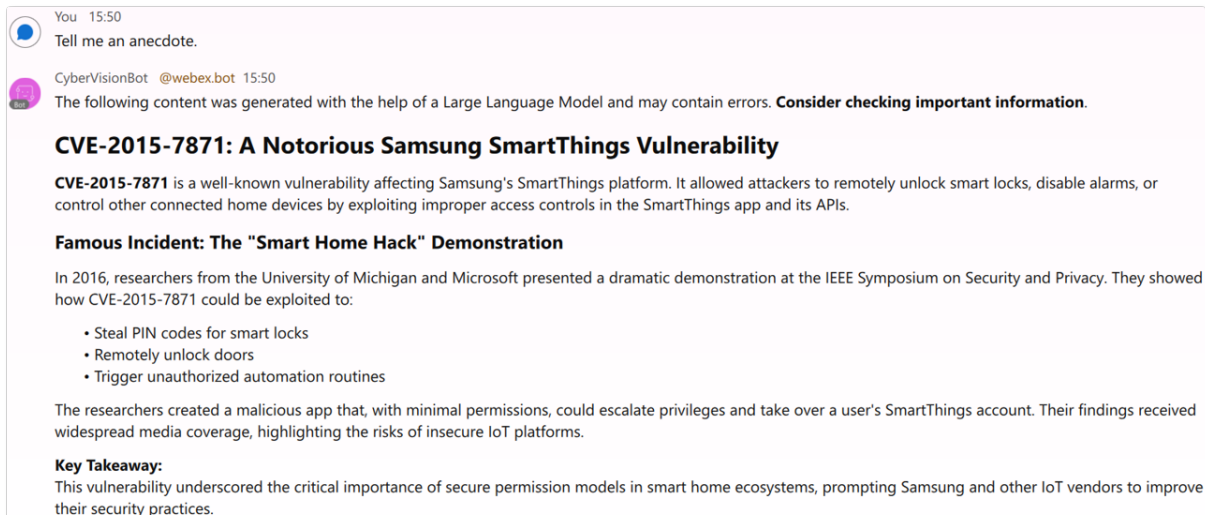


Figure 12.18. Illustrative chatbot interaction for the `vulnerability_anecdote` intent.

## 12.10 Use Cases Related to Events

Figure 12.19 shows how the chatbot supports event subscription management. The `subscribe_events` intent allows users to subscribe to system or component-related events, while the `unsubscribe_events` intent enables them to opt out of such notifications.

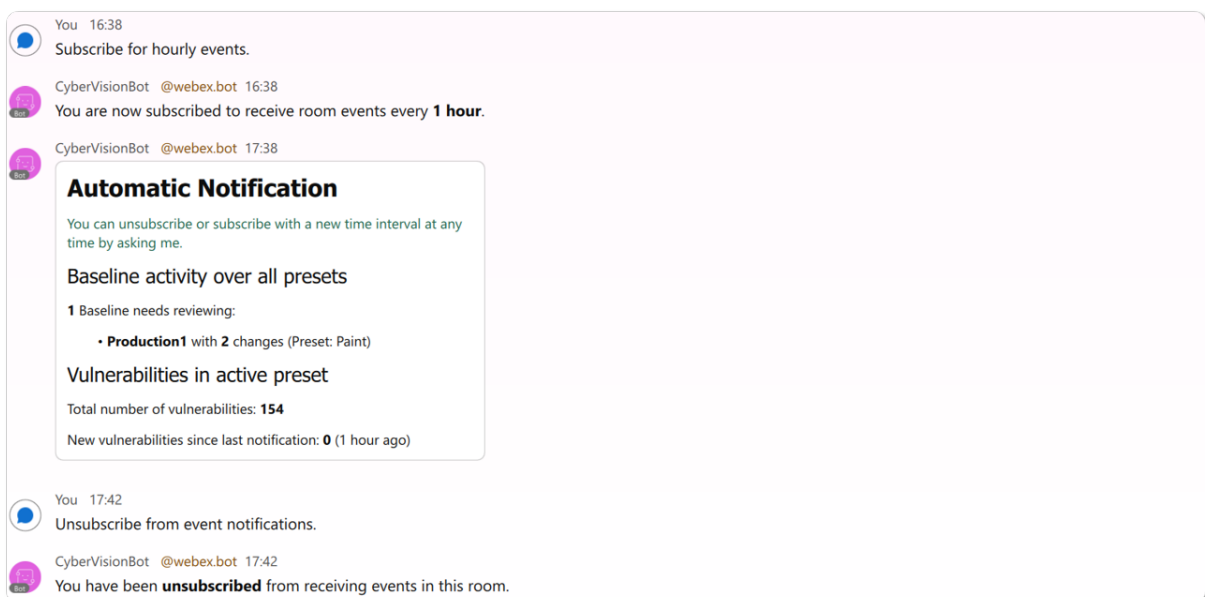


Figure 12.19. Illustrative chatbot interaction for the `subscribe_events` and `unsubscribe_events` intents.

## 12.11 Use Cases Related to Disconnection

Figure 12.20 illustrates the chatbot's handling of disconnection commands. The `disconnect` intent allows users to end their session or connection with the chatbot gracefully.

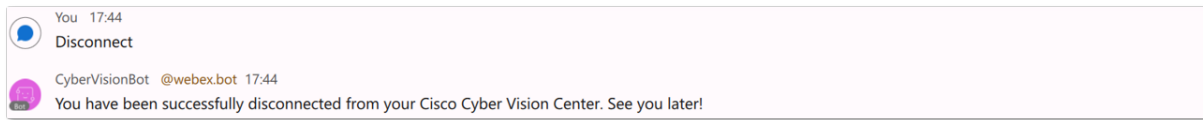


Figure 12.20. Illustrative chatbot interaction for the `disconnect` intent.

# Chapter 13

## Conclusion and Reflection

### 13.1 System Achievements and Design Insights

In this thesis, we designed and implemented a conversational AI-powered chatbot for the CCV platform, integrated with Webex, to allow users to interact with their CCV center through natural language. The development process followed several structured stages.

The initial phase involved understanding the integration environment and defining the chatbot’s technical and functional requirements. This included analyzing the CCV platform, the OT environment it supports, the Cisco Webex interface through which the chatbot is accessed, and the role of NLP in enabling conversational AI. Establishing these foundations clarified user needs, the value CCV offers, and potential use cases, thereby guiding the chatbot’s design and scope.

The goal extended beyond simply replicating existing CCV functionalities via a conversational interface. The chatbot was designed to enhance user experience with natural language interaction and to introduce new, valuable capabilities—such as explaining vulnerabilities or tags, grouping vulnerabilities by mitigation strategies, and handling non-standard inputs like typographical errors or informal phrasing. This required continuous iteration and refinement based on feedback and a deep understanding of CCV’s role in strengthening security insights for OT systems.

To address privacy concerns in Webex spaces, the chatbot uses adaptive cards—an interactive Webex utility—for authentication to the CCV center. This approach ensures confidentiality while preserving usability. Additionally, adaptive cards enabled the development of event subscription features, providing timely notifications of emerging vulnerabilities and improving user situational awareness.

The chatbot’s conversational capabilities are realized through three key components: the Interpreter, the Handler, and the Assistant.

The Interpreter is responsible for IC and ER. Two approaches were evaluated: a conventional machine learning pipeline using models like RoBERTa, CNN-LSTM, or LR, and a prompt-based LLM approach using GPT. While the LLM-based solution was ultimately chosen for the chatbot’s NLU component, it does not represent a universal improvement over conventional methods. Instead, it was selected to address specific limitations relevant

to the scope of this work—particularly the significant overhead inherent in data generation and training, processes that are highly sensitive to data quality. Moreover, the LLM approach provides greater flexibility in handling optional open-class entities (e.g., device names) and relative expressions (e.g., time references), both of which pose challenges for dataset-driven systems.

However, conventional models remain valuable, especially in resource-constrained environments where high-quality training data is available. The modular architecture of the system—separating DC, IC, and ER—enables hybrid solutions by employing simpler models where they are sufficient and reserving LLMs for tasks where they offer a clear advantage, such as ER.

The output of the Interpreter, a structured format containing the detected intent and extracted entities, is forwarded to the Handler. The Handler manages interactions with CCV’s API, ensuring queries are correctly formatted. It incorporates enhancements like fuzzy matching and context-aware parsing (e.g., country and time disambiguation) to bridge the gap between user-friendly language and strict API requirements, allowing natural interaction without compromising technical precision.

The Assistant, used by the Handler when generative responses are required, leverages an LLM with a flexible prompt structure. It is designed for scenarios where static templates or structured data are insufficient—for instance, generating explanations or performing reasoning tasks. Unlike the Interpreter, which operates under strict prompt guidelines that tightly constrain the response window, the Assistant benefits from expanded context and the addition of extension modules such as web search and the envisioned RAG system to boost accuracy and relevance.

Overall, the chatbot supports a diverse range of use cases: accessing device inventories, monitoring network activity, retrieving vulnerability insights, subscribing to event notifications, and more. The chatbot can be integrated within both direct messaging and Webex spaces, the latter of which uniquely allows for multi-user collaboration around shared information provided by the chatbot, a feature not available in the CCV web UI. Recent advances in GenAI—especially the accessibility of powerful LLMs—enable the chatbot to perform previously impractical tasks. These include inferring device origin from vendor names, interpreting relative time expressions, and generating coherent natural language explanations, all without extensive training or bespoke logic.

The system’s zero-shot IC and ER capabilities, although more computationally demanding, significantly accelerate development by eliminating the need for fine-tuning. Coupled with dynamic prompt augmentation through tools like web search, the chatbot is equipped to handle both predefined and emergent user needs with a high degree of flexibility.

## 13.2 Limitations and Future Work

While the chatbot successfully fulfills its defined functional scope (Chapter 4) and makes progress in improving technology accessibility and perceivability by enabling human-like interaction—a design goal inspired by the conversational computer of the USS Enterprise in *Star Trek*—several limitations remain. These highlight opportunities for improvement and point to future directions.

A primary limitation stems from the Handler’s design. For many use cases not delegated

to the Assistant, the Handler returns fixed, rule-based responses. Though precise and reliable for tasks such as listing devices, activities, or vulnerabilities, this rigid approach undermines the goal of conversational fluidity, resulting in outputs that may feel mechanical rather than natural.

Expanding the Assistant’s role in these scenarios could help produce more dynamic and contextual responses. However, such enhancements face practical constraints. Listing tasks often generate verbose outputs that risk exceeding current LLM context limits. More importantly, LLMs may introduce inaccuracies—a critical concern when handling sensitive data such as security vulnerabilities. Ensuring information integrity in these contexts requires a conservative, accuracy-first design, which necessarily limits the role of LLMs.

Furthermore, despite mitigation strategies, the risk of hallucination—where models generate plausible but incorrect content—remains a known limitation. This is especially critical in tasks like vulnerability explanation, where misleading or overly prescriptive advice can have unintended and potentially harmful consequences. While generative models are valuable for providing contextual insights or summarizations, their role must be carefully constrained. The Assistant should enhance understanding, not replace authoritative sources or offer definitive actions that could mislead users.

A broader design limitation lies in the chatbot’s focus on read-only interactions. The system currently supports information-retrieval use cases but deliberately omits modification capabilities. This restriction stems from a concern over ambiguity in natural-language commands, which could lead to unintended or irreversible changes to the CCV system.

As a result, users may experience a fragmented workflow—switching between the chatbot for queries and the UI for actions—thereby reducing the overall usability and coherence of the platform. Enabling safe, controlled modification capabilities through natural language remains a promising direction for future work, potentially supported by confirmation flows.

This constraint also highlights the potential for rethinking the chatbot’s integration model. Instead of Webex, integrating the chatbot directly into the CCV web UI is an alternative. This shift would eliminate the need for the chatbot to support modification use cases, as users could continue making changes directly through the existing UI. It would also unify natural-language interaction and web-based control within a single environment. In this configuration, the chatbot could expand its functionality by serving as a contextual assistant—similar to tools like GitHub Copilot [113]—supporting the user with in-place suggestions, surfacing relevant system information, and assisting with tasks based on the current interface context.

In conclusion, this thesis successfully designed and implemented a conversational AI chatbot that significantly enhances user interaction with the CCV platform. While this work marks progress in its field, it also underscores several limitations—from rigid responses and hallucination risks to functional scope and system integration—presenting important opportunities for future research and development. This project emphasized key considerations for chatbot development, explored critical design and implementation decisions, and demonstrated a successful implementation of the defined functional scope.

The chatbot’s source code is publicly available on GitHub [10], providing a robust foundation for continued research and future improvements.

# Bibliography

- [1] Jason Mars. “Breaking Bots: Inventing A New Voice In The AI Revolution”. Kindle Edition. ForbesBooks, 2021 (page 1).
- [2] Mitusha Arya. “A brief history of Chatbots”. Accessed: 2023-09-20. URL: <https://blog.chatbotslife.com/a-brief-history-of-chatbots-d5a8689cf52f> (page 1).
- [3] Court Bishop. “Chatbots vs. conversational AI: What’s the difference?” Accessed: 2023-09-20. URL: <https://www.zendesk.com/blog/chatbot-vs-conversational-ai/> (page 1).
- [4] Greg Ahern. “What is a Conversational AI Chatbot?” Accessed: 2023-09-25. URL: <https://www.ometrics.com/blog/e-commerce-chatbots/what-is-a-conversational-ai-chatbot/> (page 1).
- [5] Tulane University. “Why accessible technology is important”. Accessed: 2023-09-24. URL: <https://sopa.tulane.edu/blog/why-accessible-technology-important> (page 1).
- [6] International Telecommunication Union. “Individuals using the Internet”. Accessed: 2023-09-24. URL: <https://www.itu.int/en/itu-d/statistics/pages/stat/default.aspx> (page 2).
- [7] UW-Madison Information Technology. “What is accessible technology?” Accessed: 2023-09-24. URL: <https://it.wisc.edu/learn/make-it-accessible/what-is-accessible-technology/> (page 2).
- [8] Cambridge Dictionary. “Accessibility Definition”. Accessed: 2023-09-25. URL: <https://dictionary.cambridge.org/dictionary/english/accessibility> (page 2).
- [9] The World Wide Web Consortium. “How to Meet WCAG”. Accessed: 2023-09-25. URL: [https://www.w3.org/WAI/WCAG21/quickref/?versions=2.1&currentsidebar=%23col\\_overview](https://www.w3.org/WAI/WCAG21/quickref/?versions=2.1&currentsidebar=%23col_overview) (page 2).
- [10] Jordi Hoorelbeke. “Webex CCV Chatbot Source Code”. URL: <https://github.com/jordi-h/webex-cybervision-chatbot.git> (pages 4, 94).
- [11] Wikipedia contributors. “Cisco”. Accessed: 2023-09-20. URL: <https://en.wikipedia.org/wiki/Cisco> (page 5).
- [12] Cisco Systems. “Cisco products”. Accessed: 2023-09-25. URL: <https://www.cisco.com/c/en/us/products/index.html> (page 5).
- [13] Cisco Systems. “What Is IT (Information Technology)?” Accessed: 2024-11-24. URL: <https://www.cisco.com/c/en/us/solutions/enterprise-networks/it-information-technology-explained.html> (page 5).

- [14] Audavia. “Technologies de l’information : qu’est-ce que l’IT ? Pourquoi s’y former ?” Accessed: 2024-11-24. URL: <https://audavia-formation.fr/technologies-information/> (page 5).
- [15] Cisco Systems. “How Do OT and IT Differ?” Accessed: 2024-11-24. URL: <https://www.cisco.com/c/en/us/solutions/internet-of-things/what-is-ot-vs-it.html> (page 5).
- [16] Red Hat. “What is operational technology (OT)?” Accessed: 2024-11-24. URL: <https://www.redhat.com/en/topics/edge-computing/what-is-ot> (page 5).
- [17] “Software-Defined Networking approaches for intrusion response in Industrial Control Systems: A survey”. Accessed: 2024-11-24. URL: <https://www.sciencedirect.com/science/article/pii/S1874548223000288> (page 5).
- [18] Fortinet. “Purdue Model for ICS Security”. Accessed: 2024-11-24. URL: <https://www.fortinet.com/resources/cyberglossary/purdue-model> (pages 5, 6).
- [19] ZScaler. “What Is the Purdue Model for ICS Security?” Accessed: 2024-11-24. URL: <https://www.zscaler.com/resources/security-terms-glossary/what-is-purdue-model-ics-security> (pages 5, 7).
- [20] Cisco Systems. “Cisco Cyber Vision Data Sheet”. Accessed: 2024-11-24. URL: <https://www.cisco.com/c/en/us/products/collateral/se/internet-of-things/datasheet-c78-743222.html> (page 6).
- [21] Aveva. “SCADA vs HMI: The difference between them”. Accessed: 2024-11-25. URL: <https://www.aveva.com/en/perspectives/blog/the-difference-between-scada-and-hmi/> (page 6).
- [22] Uri Katz. “Hacking ICS Historians: The Pivot Point from IT to OT”. Accessed: 2024-11-25. URL: <https://claroty.com/team82/research/hacking-ics-historians-the-pivot-point-from-it-to-ot> (page 6).
- [23] SAP. “What is a manufacturing execution system (MES)?” Accessed: 2024-11-25. URL: <https://www.sap.com/products/scm/digital-manufacturing/what-is-mes.html> (page 6).
- [24] Cisco Dan Behrens. “Cisco Cyber Vision Overview”. Accessed: 2024-07-31. URL: <https://www.youtube.com/watch?v=0ou9i-CQ2r0> (pages 7, 8).
- [25] Cisco IoT Security Escalation team. “Cisco Cyber Vision Architecture Guide”. Accessed: 2024-11-25. URL: [https://www.cisco.com/c/dam/en/us/td/docs/security/cyber\\_vision/Cisco\\_Cyber\\_Vision\\_Architecture\\_Guide-5-0-x.pdf](https://www.cisco.com/c/dam/en/us/td/docs/security/cyber_vision/Cisco_Cyber_Vision_Architecture_Guide-5-0-x.pdf) (page 7).
- [26] Cisco Systems. “High-level architecture of Cisco Cyber Vision”. Accessed: 2024-04-28. URL: [https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/datasheet-c78-743222.docx/\\_jcr\\_content/renditions/datasheet-c78-743222\\_0.png](https://www.cisco.com/c/dam/en/us/products/collateral/se/internet-of-things/datasheet-c78-743222.docx/_jcr_content/renditions/datasheet-c78-743222_0.png) (page 8).
- [27] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Device”. Accessed: 2024-11-29. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_device.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_device.html) (page 9).



- [28] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Component”. Accessed: 2024-11-29. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_component.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_component.html) (page 9).
- [29] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Group”. Accessed: 2024-11-29. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_groups.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_groups.html) (page 10).
- [30] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Flow”. Accessed: 2024-11-29. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_flow.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_flow.html) (page 10).
- [31] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Activity”. Accessed: 2024-11-29. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_activity.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_activity.html) (page 10).
- [32] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Tag”. Accessed: 2024-11-29. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_tags.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_tags.html) (page 11).
- [33] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Vulnerability”. Accessed: 2024-11-30. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_vulnerability.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_vulnerability.html) (page 12).
- [34] NIST. “Vulnerability Metrics”. Accessed: 2024-11-30. URL: <https://nvd.nist.gov/vuln-metrics/cvss> (page 12).
- [35] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Risk score”. Accessed: 2024-11-30. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_risk-score\\_concept.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_risk-score_concept.html) (page 12).
- [36] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Preset”. Accessed: 2024-11-30. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_preset.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_preset.html) (page 12).
- [37] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 5.0.0 - Filter”. Accessed: 2024-11-30. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-5-0-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_filters.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-5-0-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_filters.html) (page 12).
- [38] Cisco Systems. “Cisco Cyber Vision GUI User Guide, Release 4.2.0 - Chapter: Monitor”. Accessed: 2024-11-30. URL: [https://www.cisco.com/c/en/us/td/docs/security/cyber\\_vision/publications/GUI/Release-4-2-0/b\\_Cisco\\_Cyber\\_Vision\\_GUI\\_User\\_Guide/m\\_monitor.html](https://www.cisco.com/c/en/us/td/docs/security/cyber_vision/publications/GUI/Release-4-2-0/b_Cisco_Cyber_Vision_GUI_User_Guide/m_monitor.html) (page 13).



- [39] Cisco Cyber Vision. “API documentation accessible inside the CCV center”. Accessed: 2024-11-30. URL: <https://developer.cisco.com/docs/cyber-vision/introduction/#cisco-cyber-vision-api-use-cases> (page 14).
- [40] Cisco Systems. “Webex Official Website”. Accessed: 2023-09-24. URL: <https://www.webex.com/> (page 14).
- [41] Cisco Systems. “Bots”. Accessed: 2023-09-25. URL: <https://developer.webex.com/docs/bots> (pages 14, 103).
- [42] Inc. Kore.ai. “Github Webex Bot”. Accessed: 2024-04-17. URL: <https://apphub.webex.com/applications/github-kore-ai-inc-5402-80591-75156> (page 14).
- [43] Cisco Systems. “Rememory Webex Bot”. Accessed: 2024-04-17. URL: <https://apphub.webex.com/applications/rememory-cisco-systems-20307> (page 14).
- [44] Cisco Systems. “Jira Cloud Webex Bot”. Accessed: 2024-04-17. URL: <https://apphub.webex.com/applications/jira-cloud-34441-68681> (page 14).
- [45] Lucid Software. “Lucid Webex Bot”. Accessed: 2024-04-17. URL: <https://apphub.webex.com/applications/lucidspark-whiteboard-lucid-software> (page 14).
- [46] Cisco Systems. “Webex Webhooks”. Accessed: 2024-04-17. URL: <https://developer.webex.com/docs/api/guides/webhooks> (page 15).
- [47] Cisco Systems. “Webhooks”. Accessed: 2024-05-02. URL: <https://developer.webex.com/docs/api/v1/webhooks> (pages 15, 24, 105).
- [48] David Galic. “What is conversational AI? How it works, examples, and more”. Accessed: 2024-12-07. URL: <https://www.zendesk.com/blog/customers-really-feel-conversational-ai/> (page 16).
- [49] IBM. “Components of conversational AI”. Accessed: 2024-12-07. URL: <https://www.ibm.com/topics/conversational-ai> (page 16).
- [50] Flask. “Flask”. Accessed: 2025-03-09. URL: <https://flask.palletsprojects.com/en/stable/> (page 23).
- [51] NGINX. “nginx”. Accessed: 2025-03-09. URL: <https://nginx.org/> (page 23).
- [52] Webex. “Get Message Details”. Accessed: 2025-03-12. URL: <https://developer.webex.com/docs/api/v1/messages/get-message-details> (page 25).
- [53] Webex. “Capacités de l’application | Webex”. Accessed: 2025-03-12. URL: <https://help.webex.com/fr-fr/article/n8vw82eb/Capacit%C3%A9s-de-l%E2%80%99application-%7C-Webex> (page 25).
- [54] Cisco. “About Adaptive Cards”. Accessed: 2025-03-23. URL: <https://developer.cisco.com/learning/modules/creating-webex-bots/webex-adaptive-cards/about-adaptive-cards/> (page 28).
- [55] Cisco Systems. “Webex’s card designer tool”. Accessed: 2024-05-02. URL: <https://developer.webex.com/buttons-and-cards-designer> (page 29).
- [56] Cisco Systems. “Cisco IoT Operations Dashboard Data Sheet”. Accessed: 2023-09-25. URL: <https://www.cisco.com/c/en/us/products/collateral/cloud-systems-management/iot-operations-dashboard/datasheet-c78-2363146.html?ccid=cc003448&oid=dstit029176&dtid=odicdc000509> (page 32).

- [57] Google. “Conversational Agents (Dialogflow CX) documentation”. Accessed: 2025-05-03. URL: <https://cloud.google.com/dialogflow/cx/docs?hl=en> (page 34).
- [58] IBM. “IBM Watson to watsonx”. Accessed: 2025-05-03. URL: <https://www.ibm.com/watson> (page 34).
- [59] Amazon (AWS). “Amazon Lex - AI Chat Builder”. Accessed: 2025-05-03. URL: [https://aws.amazon.com/lex/?nc1=h\\_ls](https://aws.amazon.com/lex/?nc1=h_ls) (page 34).
- [60] Jeff Schnurr. “Botkit Middleware Dialogflow”. Accessed: 2025-04-06. URL: <https://github.com/jschnurr/botkit-middleware-dialogflow> (page 34).
- [61] Cisco Systems. “Different Approaches for Building Conversational Applications”. Accessed: 2025-04-06. URL: [https://www.mindmeld.com/docs/intro/approaches\\_for\\_building\\_conversational\\_applications.html](https://www.mindmeld.com/docs/intro/approaches_for_building_conversational_applications.html) (pages 34, 35).
- [62] Cobus Greyling. “A Brief Overview Of Cisco MindMeld”. Accessed: 2025-04-07. URL: <https://cobusgreyling.medium.com/a-brief-overview-of-cisco-mindmeld-292430f8abc> (pages 34, 35).
- [63] Scikit-learn. “Scikit-learn”. Accessed: 2025-06-09. URL: <https://scikit-learn.org/stable/> (page 34).
- [64] TensorFlow. “TensorFlow”. Accessed: 2025-06-09. URL: <https://www.tensorflow.org/> (page 34).
- [65] PyTorch. “PyTorch”. Accessed: 2025-06-09. URL: <https://pytorch.org/> (page 34).
- [66] Hugging Face. “Transformers”. Accessed: 2025-05-04. URL: <https://huggingface.co/docs/transformers/en/index> (page 35).
- [67] Rasa. “Enterprise-grade AI, minus the black box”. Accessed: 2025-05-03. URL: <https://rasa.com/> (page 35).
- [68] Cisco Systems and GitHub Community. “MindMeld: Conversational AI Platform for Deep-Domain Voice Interfaces and Chatbots”. Accessed: 2025-05-03. URL: <https://www.mindmeld.com/> (page 35).
- [69] Cisco Systems. “Introducing MindMeld”. Accessed: 2025-04-08. URL: [https://www.mindmeld.com/docs/intro/introducing\\_mindmeld.html](https://www.mindmeld.com/docs/intro/introducing_mindmeld.html) (page 35).
- [70] Cisco Systems. “Define the Domain, Intent, Entity, and Role Hierarchy”. Accessed: 2025-04-08. URL: [https://www.mindmeld.com/docs/quickstart/03\\_define\\_the\\_hierarchy.html](https://www.mindmeld.com/docs/quickstart/03_define_the_hierarchy.html) (page 36).
- [71] Armando Murga. “Enhancing Intent Classification and Error Handling in Agentic LLM Applications”. Accessed: 2025-05-17. URL: <https://medium.com/@mr.murga/enhancing-intent-classification-and-error-handling-in-agentic-llm-applications-df2917d0a3cc> (page 36).
- [72] Sandani Sesanika Fernando. “Fine-Tuning BERT for Intent Recognition”. Accessed: 2025-04-12. URL: <https://sandanisesanika.medium.com/fine-tuning-bert-for-intent-recognition-4fef75eb5fe5> (page 38).
- [73] Cisco Systems. “Mindmeld Home Assistant blueprint”. Accessed: 2025-05-17. URL: [https://github.com/CiscoDevNet/mindmeld-blueprints/blob/develop/blueprints/home\\_assistant/domains/unknown/unknown/training.txt](https://github.com/CiscoDevNet/mindmeld-blueprints/blob/develop/blueprints/home_assistant/domains/unknown/unknown/training.txt) (page 39).

- [74] Connie K. Varnhagen et al. “lol: new language and spelling in instant messaging”. Accessed: 2025-04-13. URL: [https://www.researchgate.net/publication/225961195\\_Lol\\_New\\_language\\_and\\_spelling\\_in\\_instant\\_messaging](https://www.researchgate.net/publication/225961195_Lol_New_language_and_spelling_in_instant_messaging) (page 40).
- [75] General Architecture for Text Engineering (GATE). “Introduction to Gazetteers”. Accessed: 2025-05-03. URL: <https://gate.ac.uk/sale/tao/split.html#QQ2-18-371> (page 40).
- [76] NLTK. “Natural Language Toolkit”. Accessed: 2025-04-21. URL: <https://www.nltk.org/> (page 41).
- [77] Cisco Systems. “Working with the Text Preparation Pipeline”. Accessed: 2025-04-21. URL: [https://www.mindmeld.com/docs/userguide/text\\_preparation\\_pipeline.html](https://www.mindmeld.com/docs/userguide/text_preparation_pipeline.html) (page 41).
- [78] Cisco Systems. “Working with the Domain Classifier”. Accessed: 2025-04-12. URL: [https://www.mindmeld.com/docs/userguide/domain\\_classifier.html](https://www.mindmeld.com/docs/userguide/domain_classifier.html) (page 41).
- [79] Cisco Systems. “Working with the Entity Recognizer”. Accessed: 2025-05-03. URL: [https://www.mindmeld.com/docs/userguide/entity\\_recognizer.html](https://www.mindmeld.com/docs/userguide/entity_recognizer.html) (pages 41, 45–47).
- [80] Cisco Systems. “Deep Neural Networks in MindMeld”. Accessed: 2025-04-15. URL: [https://www.mindmeld.com/docs/userguide/deep\\_neural\\_nets.html](https://www.mindmeld.com/docs/userguide/deep_neural_nets.html) (pages 41, 47).
- [81] Cisco Systems. “Working with the Intent Classifier”. Accessed: 2025-04-22. URL: [https://www.mindmeld.com/docs/userguide/intent\\_classifier.html](https://www.mindmeld.com/docs/userguide/intent_classifier.html) (page 43).
- [82] Ashwin Ittoo. “INFO2049 Web and Text Analytics course from the University of Liège (ULiège). Chapter: Contextual embeddings”. Accessed: 2025-04-13 (pages 44, 58).
- [83] Ashwin Ittoo. “INFO2049 Web and Text Analytics course from the University of Liège (ULiège). Chapter: Attention Recap”. Accessed: 2025-04-13 (page 44).
- [84] Wikipedia contributors. “BERT (language model)”. Accessed: 2025-04-13. URL: [https://en.wikipedia.org/wiki/BERT\\_\(language\\_model\)](https://en.wikipedia.org/wiki/BERT_(language_model)) (page 44).
- [85] Gabriel Guaquire and Anna Nguyen Thanh Son. “RoBERTa vs BERT for intent classification”. Accessed: 2025-04-21. URL: <https://openreview.net/pdf?id=Nbaf-YCVRea> (page 45).
- [86] Nasser Alshammari and Saad Alanazi. “The impact of using different annotation schemes on named entity recognition”. Accessed: 2025-04-23. URL: <https://www.sciencedirect.com/science/article/pii/S1110866520301596#b0040> (page 46).
- [87] Han-Cheol Cho et al. “Named entity recognition with multiple segment representations”. Accessed: 2025-04-23. URL: <https://www.sciencedirect.com/science/article/abs/pii/S0306457313000368> (page 46).
- [88] Wikipedia contributors. “Maximum-entropy Markov model”. Accessed: 2025-04-28. URL: [https://en.wikipedia.org/wiki/Maximum-entropy\\_Markov\\_model](https://en.wikipedia.org/wiki/Maximum-entropy_Markov_model) (page 46).

- [89] Minsoo Cho et al. “Combinatorial feature embedding based on CNN and LSTM for biomedical named entity recognition”. Accessed: 2025-04-28. URL: <https://www.sciencedirect.com/science/article/pii/S1532046420300083> (page 47).
- [90] Cisco Systems. “Active Learning in MindMeld”. Accessed: 2025-05-03. URL: [https://www.mindmeld.com/docs/userguide/active\\_learning.html](https://www.mindmeld.com/docs/userguide/active_learning.html) (page 54).
- [91] Jordi Hoorelbeke. “Webex IoT OD Chatbot Source Code”. URL: <https://github.com/jordi-h/webex-iotod-chatbot.git> (page 56).
- [92] Tianzheng Troy Wang. “GPT: Origin, Theory, Application, and Future”. Accessed: 2025-05-04. URL: [https://www.cis.upenn.edu/wp-content/uploads/2021/10/Tianzheng\\_Troy\\_Wang\\_CIS498EAS499\\_Submission.pdf](https://www.cis.upenn.edu/wp-content/uploads/2021/10/Tianzheng_Troy_Wang_CIS498EAS499_Submission.pdf) (pages 58, 59).
- [93] IBM. “What is prompt engineering?” Accessed: 2025-05-11. URL: <https://www.ibm.com/think/topics/prompt-engineering> (page 59).
- [94] OpenAI. “Models”. Accessed: 2025-05-11. URL: <https://platform.openai.com/docs/models> (page 60).
- [95] OpenAI. “GPT-4.1 mini”. Accessed: 2025-05-21. URL: <https://platform.openai.com/docs/models/gpt-4.1-mini> (page 64).
- [96] Guido Appenzeller, Matt Bornstein, and Martin Casado. “Navigating the High Cost of AI Compute”. Accessed: 2025-05-24. URL: <https://a16z.com/navigating-the-high-cost-of-ai-compute/> (page 67).
- [97] Anita Kirkovska. “Llama 3 70B vs GPT-4: Comparison Analysis”. Accessed: 2025-06-08. URL: <https://www.vellum.ai/blog/llama-3-70b-vs-gpt-4-comparison-analysis> (page 67).
- [98] DAIR.AI. “Zero-Shot Prompting”. Accessed: 2025-05-24. URL: <https://www.promptingguide.ai/techniques/zeroshot> (page 67).
- [99] DAIR.AI. “Few-Shot Prompting”. Accessed: 2025-05-24. URL: <https://www.promptingguide.ai/techniques/fewshot> (page 67).
- [100] OpenAI. “Fine-tuning”. Accessed: 2025-05-24. URL: <https://platform.openai.com/docs/guides/fine-tuning> (page 68).
- [101] Rob Neuhaus and Ray Ruvinskiy. “Gibberish-Detector”. Accessed: 2025-05-24. URL: <https://github.com/rrenaud/Gibberish-Detector> (page 69).
- [102] James Padolsey. “PSA: Always sanitize LLM user inputs”. Accessed: 2025-05-24. URL: <https://padolsey.medium.com/psa-always-sanitize-llm-user-inputs-cc0a38429e98> (page 69).
- [103] Wikipedia Contributors. “Levenshtein distance”. Accessed: 2025-05-25. URL: [https://en.wikipedia.org/wiki/Levenshtein\\_distance](https://en.wikipedia.org/wiki/Levenshtein_distance) (page 75).
- [104] PyPI. “fuzzywuzzy”. Accessed: 2025-05-25. URL: <https://pypi.org/project/fuzzywuzzy/> (page 75).
- [105] WorldData.info. “International country codes”. Accessed: 2025-05-25. URL: <https://www.worlddata.info/countrycodes.php> (page 75).
- [106] PyPI. “pycountry”. Accessed: 2025-05-25. URL: <https://pypi.org/project/pycountry/> (page 75).

- [107] Kevin Doubleday. “How Decentralized GraphRAG Improves GenAI Accuracy”. Accessed: 2025-05-29. URL: <https://medium.com/fluree/how-decentralized-graphrag-improves-genai-accuracy-0cb3fd861712> (page 79).
- [108] LangChain. “Search Tools”. Accessed: 2025-05-29. URL: <https://python.langchain.com/docs/integrations/tools/> (page 80).
- [109] LangChain. “Tavily Search”. Accessed: 2025-05-29. URL: [https://python.langchain.com/docs/integrations/tools/tavily\\_search/](https://python.langchain.com/docs/integrations/tools/tavily_search/) (page 80).
- [110] NIST. “National Vulnerability Database”. Accessed: 2024-06-06. URL: <https://nvd.nist.gov/> (page 80).
- [111] Yunfan Gao et al. “Retrieval-Augmented Generation for Large Language Models: A Survey”. Accessed: 2025-06-01. URL: <https://arxiv.org/pdf/2312.10997> (pages 80, 81).
- [112] Devolutions. “DEVO-2025-0010”. Accessed: 2025-06-01. URL: <https://devolutions.net/security/advisories/DEVO-2025-0010/> (page 81).
- [113] GitHub. “GitHub Copilot”. Accessed: 2025-06-01. URL: <https://github.com/features/copilot> (page 94).

# Appendix A

## Webex Bot Creation

Cisco Webex simplifies the bot creation process, which can be initiated through its developer portal [41]. As illustrated in Figure A.1, to create a bot, one needs to provide a name, a unique username, an icon, and a description of the bot's purpose. The bot's name determines how it appears within Webex, while its username is used to invite it into either a Webex space or a direct message with a user. Additionally, the process generates an access token, which is essential for creating webhooks and enabling bot interactions.

### Bot name\*

Name of your bot as it will appear in Webex.

CyberVisionBot

### Bot username\*

The username users will use to add your bot to a space. Cannot be changed later.

ccvbot

@webex.bot

✓ ccvbot@webex.bot is available

### Icon\*





Upload your own or select from our defaults. Must be exactly 512x512px in JPEG or PNG format.



Edit

### App Hub Description\*

What does your app do, how does it benefit users, how do users get started? Does your app require a non-Webex account? If your app is not free or has additional features for paid users, please note that and link to pricing information. 1024 character limit.



## Cyber Vision Bot

The **Cyber Vision Bot** is an intelligent assistant designed to enhance your experience with [Cisco Cyber Vision](#). Powered by **LLMs**, it streamlines access to key insights from your Cyber Vision Center, helping you monitor and manage your network more efficiently.

With the Cyber Vision Bot, you can:

- Explore details about **presets, tags, baselines, devices and components, groups, sensors, activities, and vulnerabilities**.
- **Subscribe to real-time updates** on changes in your Cyber Vision Center, and customize the frequency of notifications.
- Seamlessly switch between different Cyber Vision Centers by re-authenticating when necessary.

The Cyber Vision Bot makes network monitoring intuitive and accessible, empowering you to make informed decisions with confidence.

Figure A.1. [Webex bot creation interface](#). This form collects basic bot details and generates an access token for integration. Note: The bot shown is for illustrative purposes only and may no longer be available.



# Appendix B

## Webex Bot Webhook Configuration

The setup of webhooks can be accomplished via the Webex Webhooks API [47]. Each webhook requires a *name*, a *target URL* (where messages or events are sent) and a specified *resource type*.

For the bot developed in this thesis, we use the following cURL command to establish two webhooks: one for receiving text messages and another for handling card interactions.

### Message Webhook

To receive plain text messages from Webex users or spaces, the following example cURL command creates the appropriate webhook.

```
1  curl --location 'https://webexapis.com/v1/webhooks' \  
2      --header 'Content-Type: application/json' \  
3      --header 'Authorization: Bearer <access_token>' \  
4      --data '{  
5          "name": "message-webhook",  
6          "targetUrl": "https://example/message",  
7          "resource": "messages",  
8          "event": "created"  
9      }'
```

Figure B.1. Example cURL command for creating a message webhook, with placeholder values for the access token and target URL.

### Card Webhook

To receive card interactions from Webex users or spaces, the following example cURL command is used to create the appropriate webhook.



```
1 curl --location 'https://webexapis.com/v1/webhooks' \  
2     --header 'Content-Type: application/json' \  
3     --header 'Authorization: Bearer <access_token>' \  
4     --data '{  
5         "name": "card-webhook",  
6         "targetUrl": "https://example/card",  
7         "resource": "attachmentActions",  
8         "event": "created"  
9     }'
```

Figure B.2. Example cURL command for creating a card interaction webhook, with placeholder values for the access token and target URL.

Once these webhooks are configured:

- All messages sent to the bot are forwarded to the specified message endpoint.
- Similarly, all attachment actions (i.e., card interactions) are forwarded to the designated card endpoint.

# Appendix C

## Performance Evaluation Tables

### C.1 Performance Evaluation Results for Domain Classification (Conventional NLU Interpreter)

#### C.1.1 Model Comparison Summary

Table C.1. Average Metrics for Domain Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
Decision Tree (DT)	0.6613	0.7080	0.6613	0.6518	1600
Random Forest (RF)	0.8394	0.8834	0.8394	0.8385	1600
Support Vector Machine (SVM)	0.9506	0.9544	0.9506	0.9508	1600
Logistic Regression (LR)	0.9775	0.9785	0.9775	0.9775	1600
RoBERTa	0.9994	0.9994	0.9994	0.9994	1600

#### C.1.2 RoBERTa

Table C.2. Performance Metrics - RoBERTa

Domain	Accuracy	Precision	Recall	F1-Score	Support
configuration	/	1.0000	1.0000	1.0000	160
device	/	1.0000	0.9984	0.9992	640
greeting	/	1.0000	1.0000	1.0000	480
operation	/	0.9969	1.0000	0.9984	320
<b>Avg / Total</b>	0.9994	0.9994	0.9994	0.9994	1600

Table C.3. Confusion Matrix - RoBERTa

<b>Actual / Predicted</b>	configuration	device	greeting	operation
configuration	160	0	0	0
device	0	639	0	1
greeting	0	0	480	0
operation	0	0	0	320

### C.1.3 Logistic Regression (LR)

Table C.4. Performance Metrics - Logistic Regression

<b>Domain</b>	Accuracy	Precision	Recall	F1-Score	Support
configuration	/	1.0000	0.9938	0.9969	160
device	/	0.9494	0.9969	0.9726	640
greeting	/	0.9978	0.9375	0.9667	480
operation	/	0.9969	0.9906	0.9937	320
<b>Avg / Total</b>	0.9775	0.9785	0.9775	0.9775	1600

Table C.5. Confusion Matrix - Logistic Regression

<b>Actual / Predicted</b>	configuration	device	greeting	operation
configuration	159	1	0	0
device	0	638	1	1
greeting	0	30	450	0
operation	0	3	0	317

### C.1.4 Support Vector Machine (SVM)

Table C.6. Performance Metrics - Support Vector Machine

<b>Domain</b>	Accuracy	Precision	Recall	F1-Score	Support
configuration	/	1.0000	0.9437	0.9711	160
device	/	0.8980	0.9906	0.9421	640
greeting	/	0.9840	0.8958	0.9378	480
operation	/	1.0000	0.9563	0.9776	320
<b>Avg / Total</b>	0.9506	0.9544	0.9506	0.9508	1600

Table C.7. Confusion Matrix - Support Vector Machine

<b>Actual / Predicted</b>	configuration	device	greeting	operation
configuration	151	8	1	0
device	0	634	6	0
greeting	0	50	430	0
operation	0	14	0	306

### C.1.5 Random Forest (RF)

Table C.8. Performance Metrics - Random Forest

<b>Domain</b>	Accuracy	Precision	Recall	F1-Score	Support
configuration	/	1.0000	0.8063	0.8927	160
device	/	0.7156	0.9984	0.8337	640
greeting	/	0.9904	0.6458	0.7818	480
operation	/	1.0000	0.8281	0.9060	320
<b>Avg / Total</b>	0.8394	0.8834	0.8394	0.8385	1600

Table C.9. Confusion Matrix - Random Forest

<b>Actual / Predicted</b>	configuration	device	greeting	operation
configuration	129	31	0	0
device	0	639	1	0
greeting	0	170	310	0
operation	0	53	2	265

### C.1.6 Decision Tree (DT)

Table C.10. Performance Metrics - Decision Tree

<b>Domain</b>	Accuracy	Precision	Recall	F1-Score	Support
configuration	/	1.0000	0.7937	0.8850	160
device	/	0.5688	0.8781	0.6904	640
greeting	/	0.7584	0.4708	0.5810	480
operation	/	0.7647	0.4469	0.5641	320
<b>Avg / Total</b>	0.6613	0.7080	0.6613	0.6518	1600

Table C.11. Confusion Matrix - Decision Tree

<b>Actual / Predicted</b>	configuration	device	greeting	operation
configuration	127	25	5	3
device	0	562	47	31
greeting	0	224	226	10
operation	0	157	20	143

## C.2 Performance Evaluation Results for Intent Classification (Conventional NLU Interpreter)

### C.2.1 configuration-Domained Intents

#### Model Comparison Summary

All five classification models achieved identical, perfect performance on the configuration-domained intent classification task. This is explained by the fact that there is only a single intent (`get_all_configuration_groups`) in this domain, making it impossible for the models to misclassify. Therefore, only one shared performance table and confusion matrix are presented.

Table C.12. Performance Metrics for All Models - configuration-Domained Intents

<b>Model</b>	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	1.0000	1.0000	1.0000	1.0000	160
Logistic Regression (LR)	1.0000	1.0000	1.0000	1.0000	160
Support Vector Machine (SVM)	1.0000	1.0000	1.0000	1.0000	160
Random Forest (RF)	1.0000	1.0000	1.0000	1.0000	160
Decision Tree (DT)	1.0000	1.0000	1.0000	1.0000	160

Table C.13. Confusion Matrix - Shared by All Models (configuration-Domained Intents)

<b>Actual / Predicted</b>	<code>get_all_configuration_groups</code>
<code>get_all_configuration_groups</code>	160

## C.2.2 device-Domained Intents

### Model Comparison Summary

Table C.14. Average Metrics for device-Domained Intent Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	0.9953	0.9953	0.9953	0.9953	640
Logistic Regression (LR)	0.9891	0.9891	0.9891	0.9891	640
Random Forest (RF)	0.9266	0.9287	0.9266	0.9254	640
Support Vector Machine (SVM)	0.9734	0.9737	0.9047	0.9735	640
Decision Tree (DT)	0.5422	0.6423	0.5422	0.5333	640

### RoBERTa

Table C.15. Performance Metrics - RoBERTa (device-Domained Intents)

Intent	Accuracy	Precision	Recall	F1-Score	Support
get_all_device_applications	/	0.9938	1.0000	0.9969	160
get_all_device_locations	/	0.9938	0.9938	0.9938	160
get_all_devices	/	1.0000	0.9875	0.9937	160
get_device_by_name	/	0.9938	1.0000	0.9969	160
<b>Avg / Total</b>	0.9953	0.9953	0.9953	0.9953	640

Table C.16. Confusion Matrix - RoBERTa (device-Domained Intents)

Actual / Predicted	get_all_device_applications	get_all_device_locations	get_all_devices	get_device_by_name
get_all_device_applications	160	0	0	0
get_all_device_locations	0	159	0	1
get_all_devices	1	1	158	0
get_device_by_name	0	0	0	160

### Logistic Regression (LR)

Table C.17. Performance Metrics - Logistic Regression (device-Domained Intents)

Intent	Accuracy	Precision	Recall	F1-Score	Support
get_all_device_applications	/	0.9875	0.9875	0.9875	160
get_all_device_locations	/	0.9875	0.9875	0.9875	160
get_all_devices	/	0.9812	0.9812	0.9812	160
get_device_by_name	/	1.0000	1.0000	1.0000	160
<b>Avg / Total</b>	0.9891	0.9891	0.9891	0.9891	640

Table C.18. Confusion Matrix - Logistic Regression (device-Domained Intents)

<b>Actual / Predicted</b>	get_all_device_applications	get_all_device_locations	get_all_devices	get_device_by_name
get_all_device_applications	158	1	1	0
get_all_device_locations	0	158	2	0
get_all_devices	2	1	157	0
get_device_by_name	0	0	0	160

## Support Vector Machine (SVM)

Table C.19. Performance Metrics - Support Vector Machine (device-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
get_all_device_applications	/	0.9809	0.9625	0.9716	160
get_all_device_locations	/	0.9684	0.9563	0.9623	160
get_all_devices	/	0.9455	0.9750	0.9600	160
get_device_by_name	/	1.0000	1.0000	1.0000	160
<b>Avg / Total</b>	0.9734	0.9737	0.9047	0.9735	640

Table C.20. Confusion Matrix - Support Vector Machine (device-Domained Intents)

<b>Actual / Predicted</b>	get_all_device_applications	get_all_device_locations	get_all_devices	get_device_by_name
get_all_device_applications	154	3	3	0
get_all_device_locations	1	153	6	0
get_all_devices	2	2	156	0
get_device_by_name	0	0	0	160

## Random Forest (RF)

Table C.21. Performance Metrics - Random Forest (device-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
get_all_device_applications	/	0.9222	0.9625	0.9419	160
get_all_device_locations	/	0.9080	0.9250	0.9164	160
get_all_devices	/	0.9704	0.8187	0.8881	160
get_device_by_name	/	0.9143	1.0000	0.9552	160
<b>Avg / Total</b>	0.9266	0.9287	0.9266	0.9254	640

Table C.22. Confusion Matrix - Random Forest (device-Domained Intents)

<b>Actual / Predicted</b>	get_all_device_applications	get_all_device_locations	get_all_devices	get_device_by_name
get_all_device_applications	154	2	1	3
get_all_device_locations	3	148	3	6
get_all_devices	10	13	131	6
get_device_by_name	0	0	0	160

## Decision Tree (DT)

Table C.23. Performance Metrics - Decision Tree (device-Domained Intents)

Intent	Accuracy	Precision	Recall	F1-Score	Support
get_all_device_applications	/	0.7683	0.3937	0.5207	160
get_all_device_locations	/	0.7228	0.4562	0.5594	160
get_all_devices	/	0.6667	0.3750	0.4800	160
get_device_by_name	/	0.4114	0.9437	0.5731	160
<b>Avg / Total</b>	0.5422	0.6423	0.5422	0.5333	640

Table C.24. Confusion Matrix - Decision Tree (device-Domained Intents)

Actual / Predicted	get_all_device_applications	get_all_device_locations	get_all_devices	get_device_by_name
get_all_device_applications	63	12	14	71
get_all_device_locations	8	73	16	63
get_all_devices	11	7	60	82
get_device_by_name	0	9	0	151

## C.2.3 greeting-Domained Intents

### Model Comparison Summary

Table C.25. Average Metrics for greeting-Domained Intent Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	0.9854	0.9856	0.9854	0.9854	480
Logistic Regression (LR)	0.9438	0.9456	0.9437	0.9440	480
Random Forest (RF)	0.9000	0.9020	0.9000	0.9004	480
Support Vector Machine (SVM)	0.9146	0.9193	0.9146	0.9151	480
Decision Tree (DT)	0.6354	0.7229	0.6354	0.6339	480

## RoBERTa

Table C.26. Performance Metrics - RoBERTa (greeting-Domained Intents)

Intent	Accuracy	Precision	Recall	F1-Score	Support
exit	/	0.9936	0.9688	0.9810	160
greet	/	0.9695	0.9938	0.9815	160
help	/	0.9938	0.9938	0.9938	160
<b>Avg / Total</b>	0.9854	0.9856	0.9854	0.9854	480



Table C.27. Confusion Matrix - RoBERTa (greeting-Domained Intents)

<b>Actual / Predicted</b>	exit	greet	help
exit	155	4	1
greet	1	159	0
help	0	1	159

## Logistic Regression (LR)

Table C.28. Performance Metrics - Logistic Regression (greeting-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
exit	/	0.9494	0.9375	0.9434	160
greet	/	0.9006	0.9625	0.9305	160
help	/	0.9868	0.9313	0.9582	160
<b>Avg / Total</b>	0.9438	0.9456	0.9437	0.9440	480

Table C.29. Confusion Matrix - Logistic Regression (greeting-Domained Intents)

<b>Actual / Predicted</b>	exit	greet	help
exit	150	10	0
greet	4	154	2
help	4	7	149

## Support Vector Machine (SVM)

Table C.30. Performance Metrics - Support Vector Machine (greeting-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
exit	/	0.8644	0.9563	0.9080	160
greet	/	0.9006	0.9062	0.9034	160
help	/	0.9930	0.8812	0.9338	160
<b>Avg / Total</b>	0.9146	0.9193	0.9146	0.9151	480

Table C.31. Confusion Matrix - Support Vector Machine (greeting-Domained Intents)

<b>Actual / Predicted</b>	exit	greet	help
exit	153	7	0
greet	14	145	1
help	10	9	141

## Random Forest (RF)

Table C.32. Performance Metrics - Random Forest (greeting-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
exit	/	0.8547	0.9187	0.8855	160
greet	/	0.8974	0.8750	0.8861	160
help	/	0.9539	0.9062	0.9295	160
<b>Avg / Total</b>	0.9000	0.9020	0.9000	0.9004	480

Table C.33. Confusion Matrix - Random Forest (greeting-Domained Intents)

<b>Actual / Predicted</b>	exit	greet	help
exit	147	9	4
greet	17	140	3
help	8	7	145

## Decision Tree (DT)

Table C.34. Performance Metrics - Decision Tree (greeting-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
exit	/	0.4982	0.8500	0.6282	160
greet	/	0.9091	0.4375	0.5907	160
help	/	0.7615	0.6188	0.6828	160
<b>Avg / Total</b>	0.6354	0.7229	0.6354	0.6339	480

Table C.35. Confusion Matrix - Decision Tree (greeting-Domained Intents)

<b>Actual / Predicted</b>	exit	greet	help
exit	136	5	19
greet	78	70	12
help	59	2	99

## C.2.4 operation-Domained Intents

### Model Comparison Summary

Table C.36. Average Metrics for operation-Domained Intent Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	1.0000	1.0000	1.0000	1.0000	320
Logistic Regression (LR)	1.0000	1.0000	1.0000	1.0000	320
Random Forest (RF)	0.9906	0.9906	0.9906	0.9906	320
Support Vector Machine (SVM)	1.0000	1.0000	1.0000	1.0000	320
Decision Tree (DT)	0.9031	0.9132	0.9031	0.9025	320

### RoBERTa

Table C.37. Performance Metrics - RoBERTa (operation-Domained Intents)

Intent	Accuracy	Precision	Recall	F1-Score	Support
get_all_alerts	/	1.0000	1.0000	1.0000	160
get_all_events	/	1.0000	1.0000	1.0000	160
<b>Avg / Total</b>	1.0000	1.0000	1.0000	1.0000	320

Table C.38. Confusion Matrix - RoBERTa (operation-Domained Intents)

<b>Actual / Predicted</b>	get_all_alerts	get_all_events
get_all_alerts	160	0
get_all_events	0	160

### Logistic Regression (LR)

Table C.39. Performance Metrics - Logistic Regression (operation-Domained Intents)

Intent	Accuracy	Precision	Recall	F1-Score	Support
get_all_alerts	/	1.0000	1.0000	1.0000	160
get_all_events	/	1.0000	1.0000	1.0000	160
<b>Avg / Total</b>	1.0000	1.0000	1.0000	1.0000	320

Table C.40. Confusion Matrix - Logistic Regression (operation-Domained Intents)

<b>Actual / Predicted</b>	get_all_alerts	get_all_events
get_all_alerts	160	0
get_all_events	0	160

## Support Vector Machine (SVM)

Table C.41. Performance Metrics - Support Vector Machine (operation-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
get_all_alerts	/	1.0000	1.0000	1.0000	160
get_all_events	/	1.0000	1.0000	1.0000	160
<b>Avg / Total</b>	1.0000	1.0000	1.0000	1.0000	320

Table C.42. Confusion Matrix - Support Vector Machine (operation-Domained Intents)

<b>Actual / Predicted</b>	get_all_alerts	get_all_events
get_all_alerts	160	0
get_all_events	0	160

## Random Forest (RF)

Table C.43. Performance Metrics - Random Forest (operation-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
get_all_alerts	/	0.9937	0.9875	0.9906	160
get_all_events	/	0.9876	0.9938	0.9907	160
<b>Avg / Total</b>	0.9906	0.9906	0.9906	0.9906	320

Table C.44. Confusion Matrix - Random Forest (operation-Domained Intents)

<b>Actual / Predicted</b>	get_all_alerts	get_all_events
get_all_alerts	158	2
get_all_events	1	159

## Decision Tree (DT)

Table C.45. Performance Metrics - Decision Tree (operation-Domained Intents)

<b>Intent</b>	Accuracy	Precision	Recall	F1-Score	Support
get_all_alerts	/	0.9778	0.8250	0.8949	160
get_all_events	/	0.8486	0.9812	0.9101	160
<b>Avg / Total</b>	0.9031	0.9132	0.9031	0.9025	320

Table C.46. Confusion Matrix - Decision Tree (operation-Domained Intents)

<b>Actual / Predicted</b>	get_all_alerts	get_all_events
get_all_alerts	132	28
get_all_events	3	157

## C.3 Performance Evaluation Results for Entity Recognition (Conventional NLU Interpreter)

### C.3.1 Per-Configuration Average F1-Scores for Entity Recognition Models

Table C.47. Per-Configuration Average F1-Scores for Entity Recognition Models

<b>Model</b>	random-random	random-realistic	realistic-random	realistic-realistic	mix-random	mix-realistic
MEMM	0.988	0.988	0.988	0.988	0.988	0.988
LSTM	0.991	0.991	0.991	0.991	0.990	0.990
CNN-LSTM	0.998	0.994	0.994	0.995	0.992	0.991
BERT	0.993	0.941	1.000	0.997	0.997	0.988

### C.3.2 Train: Random Device Names, Test: Random Device Names

Table C.48. Average Metrics for Entity Recognition Models (Train: random device names, Test: random device names)

<b>Model</b>	Accuracy	Precision	Recall	F1-Score
MEMM	0.988	0.988	0.988	0.988
LSTM	0.991	0.991	0.991	0.991
CNN-LSTM	0.998	0.998	0.998	0.998
BERT	0.993	0.993	0.993	0.993

### C.3.3 Train: Random Device Names, Test: Realistic Device Names

Table C.49. Average Metrics for Entity Recognition Models (Train: random device names, Test: realistic device names)

<b>Model</b>	Accuracy	Precision	Recall	F1-Score
MEMM	0.988	0.988	0.988	0.988
LSTM	0.991	0.991	0.991	0.991
CNN-LSTM	0.994	0.994	0.994	0.994
BERT	0.945	0.945	0.945	0.941

### C.3.4 Train: Realistic Device Names, Test: Random Device Names

Table C.50. Average Metrics for Entity Recognition Models (Train: realistic device names, Test: random device names)

Model	Accuracy	Precision	Recall	F1-Score
MEMM	0.988	0.988	0.988	0.988
LSTM	0.990	0.990	0.990	0.991
CNN-LSTM	0.994	0.994	0.994	0.994
BERT	1.000	1.000	1.000	1.000

### C.3.5 Train: Realistic Device Names, Test: Realistic Device Names

Table C.51. Average Metrics for Entity Recognition Models (Train: realistic device names, Test: realistic device names)

Model	Accuracy	Precision	Recall	F1-Score
MEMM	0.988	0.988	0.988	0.988
LSTM	0.990	0.990	0.990	0.991
CNN-LSTM	0.995	0.995	0.995	0.995
BERT	0.997	0.997	0.997	0.997

### C.3.6 Train: Mixed Device Names, Test: Random Device Names

Table C.52. Average Metrics for Entity Recognition Models (Train: mixed device names, Test: random device names)

Model	Accuracy	Precision	Recall	F1-Score
MEMM	0.988	0.988	0.988	0.988
LSTM	0.990	0.990	0.990	0.990
CNN-LSTM	0.992	0.992	0.992	0.992
BERT	0.997	0.997	0.997	0.997

### C.3.7 Train: Mixed Device Names, Test: Realistic Device Names

Table C.53. Average Metrics for Entity Recognition Models (Train: mixed device names, Test: realistic device names)

Model	Accuracy	Precision	Recall	F1-Score
MEMM	0.988	0.988	0.988	0.988
LSTM	0.990	0.990	0.990	0.990
CNN-LSTM	0.990	0.990	0.990	0.991
BERT	0.988	0.988	0.988	0.988

## C.4 Performance Evaluation Results for Intent Classification (RoBERTa vs. GPT-4.1-mini-2025-04-14 Comparison)

### C.4.1 configuration-Domained Intents

Table C.54. Average Metrics for configuration-Domained Intent Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	1.0000	1.0000	1.0000	1.0000	160
GPT-4.1-mini-2025-04-14	1.0000	1.0000	1.0000	1.0000	160

### C.4.2 device-Domained Intents

Table C.55. Average Metrics for device-Domained Intent Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	0.9953	0.9953	0.9953	0.9953	640
GPT-4.1-mini-2025-04-14	0.9875	0.9908	0.9875	0.9891	640

### C.4.3 greeting-Domained Intents

Table C.56. Average Metrics for greeting-Domained Intent Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	0.9854	0.9856	0.9854	0.9854	480
GPT-4.1-mini-2025-04-14	0.9979	0.9979	0.9979	0.9979	480

#### C.4.4 operation-Domained Intents

Table C.57. Average Metrics for operation-Domained Intent Classification Models

Model	Accuracy	Precision	Recall	F1-Score	Support
RoBERTa	1.0000	1.0000	1.0000	1.0000	320
GPT-4.1-mini-2025-04-14	1.0000	1.0000	1.0000	1.0000	320

### C.5 Performance Evaluation Results for Intent Classification and Entity Recognition on CCV Use Cases (LLM-Based NLU Interpreter)

Table C.58. Intent Classification and Entity Recognition on CCV Use Cases - Performance Summary

Metric	Value
Total entries evaluated	520
Correct intents	517
Correct entities	517
Global IC Accuracy	0.9942
Global IC Precision	0.9912
Global IC Recall	0.9942
Global IC F1-Score	0.9927
Average ER Accuracy	0.9942

### C.6 Robustness Evaluation Results for greet Intent Classification (Noisy Test Sets)

Table C.59. Per-Model Average Recall for greet Intent Classification with Noisy Test Datasets

Noise Level	Recall (SVM)	Recall (LR)	Recall (RoBERTa)	Recall (GPT-4.1-mini-2025-04-14)
0.00	0.9062	0.9625	0.9938	1.0000
0.05	0.7812	0.9313	0.9625	0.9875
0.10	0.7625	0.9437	0.9812	1.0000
0.15	0.6937	0.9125	0.9500	0.9875
0.20	0.6438	0.9062	0.9500	0.9812
0.25	0.5250	0.8875	0.9625	0.9875