Developing a Security Chatbot/Assistant with LLM Tools - Universal Chatbot

Objective Definition

A security chatbot should serve as a 24/7 virtual security consultant, offering personalized risk assessments, proactive recommendations, and reliable incident handling while maintaining strict security standards. Its multi-LLM capability, scalability, and compliance make it a valuable tool in physical security, smart home protection, and enterprise security solutions.

1. Requirements

1.1 User Interaction & Response Capabilities

- Provide security recommendations based on user input (e.g., home security, business premises, event security).
- Answer security-related queries (e.g., “What is the best surveillance system for a warehouse?”).
- Adapt responses based on user profile and security needs.
- Support real-time, multi-turn chatbot/assistant conversations to refine recommendations based on user feedback.

1.2 Multi-LLM Integration & Model Switching

- Ability to switch between multiple LLM Services and Models (e.g., xAI Grok 2, OpenAI GPT-4o, Cohere Command-R) to compare insights.
- Dynamically select the best model based on query type (e.g., xAI Grok 2 for high-level advice, OpenAI GPT-4o for in-depth analysis).

1.3 Security Assessment Capabilities

- Perform risk assessment based on user-provided parameters (e.g., property size, location, security concerns).
- Recommend security hardware (e.g., cameras, motion sensors, alarms, access control systems).
- Suggest best practices for different environments (e.g., home, corporate office, warehouse, event security).
- Identify potential security vulnerabilities based on user descriptions.

1.4 Integration with External Security Services (Optional for Advanced Versions)

- Pull data from security databases (e.g., crime rates, local security threats).
- Integrate with IoT security devices (e.g., smart locks, surveillance cameras) to offer real-time monitoring insights.
- Provide emergency contacts or recommend security agencies based on location.

1.5 Incident Reporting & Logging

- Allow users to report security incidents (e.g., break-ins, suspicious activities).
- Maintain a log of past interactions and recommendations for future reference.
- Generate incident reports that users can share with law enforcement or security personnel.

2. Development Process

The universal chatbot was built in Python, leveraging its robust ecosystem, and integrates three LLMs: Grok 2 API for innovative responses, OpenAI GPT-4o API for depth, and Cohere Command-R API for efficiency. Libraries such as `requests`, `tenacity`, and `dotenv` supported API calls, retry mechanisms, and key management. Prompts like “Act as a physical security consultant” shaped the chatbot’s responses, with Grok optionally using “DeepSearch” to analyze trends. Unlike Poe’s prompt-only interface or Dialogflow’s visual flow builder, Python required manual coding but offered unparalleled customization. The application was tested using a `unittest` suite with mocked APIs, ensuring reliability and functionality.
When I started this journey deciding what type of chatbot I would build; I decided that for this assignment I would look at 3 alternative platforms: 1) Poe by Quora, 2) DialogFlow by Google and 3) building a custom universal chatbot in Python against the actual LLM APIs, With a could take advantage of and switch between multiple LLM services and models.The Universal Chatbot, developed in Python, by Michael P Williams, integrates xAI’s Grok 2, OpenAI’s GPT-4o, and Cohere’s Command-R into a single chat interface, This contrasts with chatbots I built on Poe (`mpwsecuritybot`) and Google Dialogflow. Poe, created by Quora, offers a user-friendly platform aggregating multiple Large Language Models (LLMs) such as ChatGPT and Claude, with a simple web interface for creating custom bots like `mpwsecuritybot`, a security consultant. It requires minimal coding, relying on prompt-based training, but lacks deep customization and local control. Google Dialogflow, a Google-backed NLP platform, excels in intent recognition and entity extraction, enabling structured conversational flows (e.g., a security Q&A bot). It integrates seamlessly with Google services but demands more setup for complex logic and external API calls compared to Python’s flexibility. In contrast, the Python-based Universal Chatbot offers full programmatic control, multi-LLM switching, and local testing capabilities, though it requires significant coding effort unlike Poe’s simplicity or Dialogflow’s pre-built tools.

2.1 Criteria for Selection

These three alternatives were chosen based on their varying levels of ease of development, flexibility, scalability, and control over responses. Poe offers a fast and simple interface, Dialogflow enables structured intent-based flows, and a custom Python chatbot allows full programmatic control. Evaluating these approaches provided insight into the trade-offs between convenience and customization in chatbot development.
The Universal Chatbot, developed in Python by Michael P. Williams, integrates xAI's Grok 2, OpenAI's GPT-4o, and Cohere's Command-R into a single chat interface. This contrasts with chatbots I built on Poe (mpwsecuritybot) and Google Dialogflow.

2.2 Comparison of Development Effort

- **Poe (Low Effort)** – Requires minimal coding, relying on prompt-based training. It allows for quick bot creation but lacks deep customization and local control.
- **Dialogflow (Medium Effort)** – Demands more setup for API calls, entity recognition, and structured flows. It is highly effective for predefined conversational paths but struggles with unstructured queries.
- **Universal Chatbot (High Effort)** – Requires full development effort but provides ultimate control, multi-LLM switching, local testing, and deep customization.

2.3 Limitations of Poe and Dialogflow

Poe's Constraints:

- Limited to predefined models (ChatGPT, Claude, etc.)
- No ability to customize retrieval methods or integrate proprietary LLMs
- No local control or ability to modify model behavior at runtime

Dialogflow's Constraints:

- Rigid intent-based design makes it less adaptable for open-ended discussions
- Extensive setup is required for API calls, authentication, and dynamic responses
- Less flexible for real-time LLM switching or comparing different model outputs

2.4 Unique Advantages of the Universal Security Chatbot in Python

- **Multi-LLM Switching** – Unlike Poe and Dialogflow, which force you to commit to a single model, the Universal Chatbot allows dynamic switching between Grok-2, GPT-4o, and Command-R.
- **Local Testing & Deployment** – The chatbot can be tested and deployed flexibly without reliance on a third-party cloud platform.

- **Customization & Control** – Enables fine-tuned prompt engineering, API rate limiting, error handling, and response formatting, which are limited or unavailable in Poe and Dialogflow.

3. Business & Real-World Applications

- **Poe** – Best for quick bot experiments and individual use but lacks enterprise-level integration.
- **Dialogflow** – Ideal for customer service bots with predefined intent-based interactions but requires significant setup for complex use cases.
- **Universal Chatbot** – Best suited for businesses requiring deep AI integration, such as physical security consultation, where responses need to be dynamically tailored and enhanced across multiple LLMs.

4. Performance Evaluation

Testing revealed Grok's concise advice, GPT-4o's detailed and contextual replies, and Command-R's efficiency, meeting expectations for varied inputs. Successful interactions included mid-conversation LLM switching, outperforming Poe's static model choice and Dialogflow's rigid flows. Challenges included initial syntax errors (resolved by removing a stale `site-packages` file) and Grok's error handling, addressed by initializing `resp_grok`. Poe's `mpwsecuritybot` was simpler but less dynamic, while Dialogflow struggled with unstructured queries compared to the Python chatbot's adaptability.

5. Business Application

In security consulting or smart home industries, the Universal Chatbot could provide 24/7 assessments, recommend products (e.g., smart locks), and analyze trends, surpassing Poe's basic Q&A capabilities and Dialogflow's limited scalability. Its multi-LLM capability adds value by tailoring responses to client needs, making it ideal for customer engagement or upselling hardware in a professional setting.

6. Reflective Summary

The chatbot is business-ready due to its flexibility and robust performance, though API reliability and error handling require refinement—advantages over Poe's simplicity and Dialogflow's constraints. Development was smooth with Python's tools but challenging due to import conflicts and LLM-specific tweaks (e.g., Cohere's role adjustments). Key lessons include the importance of modularity for scalability, the power of multi-LLM integration compared to single-model

platforms like Poe or Dialogflow, and the necessity of rigorous testing—insights that will guide future AI projects toward greater resilience and adaptability.

I. References:

1. xAI. (2025). *xAI API Documentation*. Retrieved from https://api.x.ai/v1/chat/completions
2. OpenAI. (2025). *OpenAI API Reference*. Retrieved from https://platform.openai.com/docs/api-reference
3. Cohere. (2025). *Cohere API Documentation*. Retrieved from https://docs.cohere.ai/docs
4. Poe. (2025). *Poe Platform Documentation*. Retrieved from https://poe.com/help
5. Google. (2025). *Dialogflow Official Documentation*. Retrieved from https://cloud.google.com/dialogflow/docs
6. Python Software Foundation. (2025). *Python 3.12 Documentation*. Retrieved from https://docs.python.org/3.12/
7. Reitz, K. (2025). *Requests: HTTP for Humans*. Retrieved from https://requests.readthedocs.io/en/latest/
8. Bader, J. (2025). *Tenacity Documentation*. Retrieved from https://tenacity.readthedocs.io/en/latest/
9. Thakur, S. (2025). *Python-Dotenv Documentation*. Retrieved from https://github.com/theskumar/python-dotenv
10. Python Software Foundation. (2025). *unittest Documentation*. Retrieved from https://docs.python.org/3.12/library/unittest.html
11. Python Software Foundation. (2025). *Python 3.12*. Retrieved from https://www.python.org/downloads/release/python-3120/
12. Reitz, K. (2025). *Requests Library (Version 2.31.0)*. Retrieved from https://requests.readthedocs.io/en/latest/
13. Bader, J. (2025). *Tenacity Library (Version 8.2.3)*. Retrieved from https://tenacity.readthedocs.io/en/latest/
14. Thakur, S. (2025). *Python-Dotenv (Version 1.0.1)*. Retrieved from https://github.com/theskumar/python-dotenv
15. OpenAI. (2025). *OpenAI SDK (Version 1.3.0)*. Retrieved from https://platform.openai.com/docs/api-reference
16. Cohere. (2025). *Cohere SDK (Version 5.0.0)*. Retrieved from https://docs.cohere.ai/docs
17. Poe. (2025). *Poe Platform Documentation*. Retrieved from https://poe.com/
18. Google. (2025). *Dialogflow Official Documentation*. Retrieved from https://cloud.google.com/dialogflow
19. Python Software Foundation. (2025). *unittest Library Documentation*. Retrieved from https://docs.python.org/3.12/library/unittest.html
20. Python Software Foundation. (2025). *Virtualenv (Version 20.26.2)*. Retrieved from https://virtualenv.pypa.io/en/latest/
21. JetBrains. (2025). *PyCharm (Version 2023.3)*. Retrieved from https://www.jetbrains.com/pycharm/
22. Git Project. (2025). *Git (Version 2.43.0)*. Retrieved from https://git-scm.com/

II. List of LLM Services and Models Used:

The Universal Chatbot is designed to support multiple Large Language Models (LLMs) from different providers, ensuring flexibility and comparative analysis. The chatbot's architecture allows for seamless switching between models and the potential integration of additional providers in the future. Below is a structured breakdown of the LLM services and models currently implemented.

---

1. xAI Models

- **Grok-2** – The primary model from xAI, optimized for concise and innovative responses. It serves as the default model for queries requiring direct and efficient answers.
- **Grok-2-mini** – A hypothetical smaller variant, assumed to be available by February 2025. Designed for efficiency testing, this model would potentially provide faster, lower-cost responses with reduced computational overhead.

---

2. OpenAI Models

- **GPT-4o** – OpenAI's advanced multimodal model, known for its high contextual awareness and comprehensive responses. It serves as the default model for the OpenAI service.
- **GPT-3.5-turbo** – A widely used model, optimized for efficiency and cost-effectiveness. Integrated to provide a balance between speed and accuracy for general-purpose interactions.
- **GPT-4-turbo** – A hypothetical high-performance variant, assumed to be available in early 2025. Included for performance benchmarking and comparative analysis with other models.

---

3. Cohere Models

- **Command-R** – Cohere's default chat model, offering a balance of speed, quality, and reliability. Primarily used for structured and efficient responses in conversational AI tasks.
- **Command** – A lighter version of Cohere's chat models, included for versatility and cost-sensitive applications.

---

4. Potential Future Model Integrations

The chatbot's framework is **designed for expansion**, enabling the integration of additional LLM providers based on API availability and business needs.

Anthropic Claude Models *(commented out, requires SDK integration)*

- **Claude-3-opus** – Hypothetical high-capacity model designed for advanced reasoning and detailed responses.
- **Claude-3-sonnet** – Hypothetical balanced model optimized for speed and quality trade-offs.

Google Gemini Models *(subject to API access availability)*

- **Gemini AI** models could be integrated in a similar service handler to expand chatbot capabilities, particularly in Google service-oriented environments.