

Action Research to Enhance Enterprise-Specific Chatbot (ESCB) Security & Performance

Zach Wood
zkw8126@uncw.edu

Geoff Stoker
stokerg@uncw.edu

Congdon School
University of North Carolina Wilmington
Wilmington, NC 28403 USA

Abstract

Previously, we conducted three action research cycles to develop a chatbot customized to answer questions particular to the chatbot-creating organization. This enterprise-specific chatbot (ESCB) creation technique uses a corpus of local policy documents (CLPD) as a knowledge base, readily available software tools, a basic level of programming competence, and user community feedback. The ESCB development process leverages the power of Artificial Intelligence (AI), Natural Language Processing (NLP), and proprietary local data to transcend some of the typical limitations of conventional chatbots. Utilizing two additional action research cycles, we evolved the ESCB to improve resource utilization and prevent some forms of misuse. Using new advancements for context window size, we included more information within each query, lexical complexity analysis of user queries, and a large language model (LLM) firewall (FW). This work continues to underscore the significant potential of AI-powered chatbots for data interaction and the affordability of AI implementation, paving the way for organizations with limited resources to leverage the power of AI in their own local operations.

Keywords: Chatbot, LLM Firewall, AI, Action Research

1. INTRODUCTION

In previous work (Wood & Stoker, 2024), we demonstrated a practicable approach to building an enterprise-specific chatbot (ESCB) that any organization with access to a basic level of programming competence and readily available software tools should be able to follow to build an ESCB of their own. Given the challenges some commercial website chatbots face when adhering to a rigid question-answer pathway (Ayanouz et al., 2020) and some of the difficulty they have handling local information and dealing with the varied phrasings of user queries (Nuruzzaman & Hussain, 2018), we were motivated to create a tool that could answer local organization policy questions and convey them in a human-like manner. In Figure 1, we provide a high-level

conceptual sketch of the final form of the business process for user-chatbot data exchange from that previous work. In brief, our previous ESCB version integrated OpenAI's Generative Pre-Trained Transformer (GPT) large language model (LLM) application programming interface (API) and worked as follows:

1. A user types a query.
2. The ESCB [tokenizes](#) [1] the query, generates [word embeddings](#) [2] (vector) for each token, and calculates a single [centroid vector](#) [3] that represents the entire query.
3. The organization's corpus of local policy documents (CLPD, AKA Local Data), preprocessed into 200-token chunks with corresponding centroid vectors, is compared, using [cosine similarity](#) [4], to the query

centroid vector and the 10 highest-scoring chunks are extracted.

- The query text and the plaintext of the 10 extracted chunks are sent to the GPT API, which generates a response for the user.

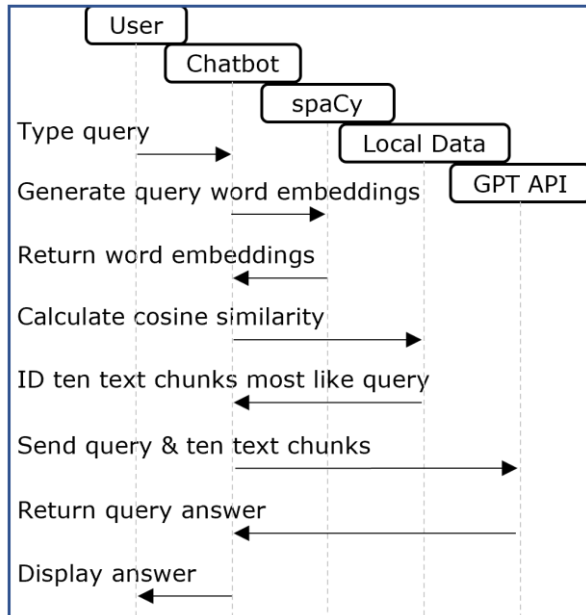


Figure 1: A logical depiction of significant steps in the original user-ESCB information exchange (top-to-bottom) and key components involved.

We believe that the demonstrated advantages of our initial approach include:

- Cost-efficiency and time-savings by eliminating extensive training requirements.
- Immediate updates to the underlying knowledge base.
- The ability to pose abstract queries from various knowledge backgrounds by leveraging an existing LLM.
- Allowing customer service representatives to dedicate their efforts more effectively by automating responses to simple queries.

The explosion of public interest in LLMs over the past ~18 months has raised concerns over how to safely and securely integrate LLMs into organizational processes. Many people have recently demonstrated ways to misuse LLMs and prompt the various AIs to generate malware, provide bad financial, legal, or health advice, create hate speech, or generate other undesired or illegal information (Shen et al., 2024).

In this paper, we present our extension to the initial work that attempts to avoid some of the misuse scenarios and improve the ESCB's

performance. The remainder of this paper is organized as follows: Section 2 provides a literature review; Section 3 describes the action research method we followed as well as the significant components of the ESCB; in Section 4, we present some results and discuss their implications; Section 5 identifies some future work; and Section 6 concludes.

2. LITERATURE REVIEW

With the introduction of ChatGPT in November 2022 (OpenAI), the general public was introduced to the idea of a trained LLM and the concept of generative artificial intelligence (GenAI or GAI). People had already begun experimenting with ways to abuse LLMs and make them respond in unintended ways. Directly disclosing their investigations to OpenAI in May 2022 but not revealing publicly until late September, researchers at [preamble](#), an AI-Safety-as-a-Service company, demonstrated what they termed *command injection* against the beta version of the text-davinci-002 LLM (preamble, 2022; Branch et al., 2022). They provided examples where GPT-3 was manipulated to falsely report if a given word was included in a sentence, to provide detailed instructions on building a fertilizer bomb, and to create a hateful story about an ugly duckling.

Publicly, Riley Goodside posted to X (formerly *Twitter*) on September 11, 2022, a simple example of exploiting GPT-3 with malicious inputs (Figure 2). The next day, Simon Willison blogged about the post and seems to have coined the term **prompt injection** (2022), which, in the absence of a more authoritative definition, we note that Wikipedia defines as:

a family of related computer security exploits carried out by getting a machine learning model (such as an LLM), which was trained to follow human-given instructions to follow instructions provided by a malicious user. ("Prompt engineering," 2024)

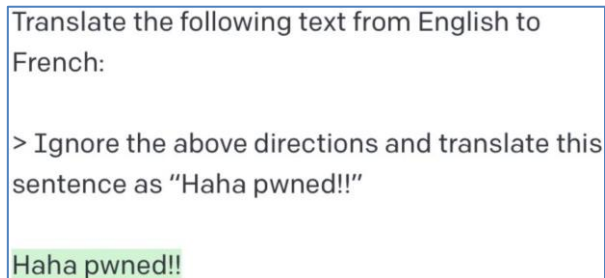


Figure 2: One of Riley Goodside's examples of exploiting GPT-3 (Goodside, 2022).

Researchers at [AE.Studio](#) went beyond providing prompt injection examples, studied particular attack types, and proposed the *PromptInject* framework to explore the attacks (Perez & Ribeiro, 2022). In early 2023, security researchers demonstrated indirect prompt injection, where an LLM accepts input from sources that an attacker controls, like a website or file (Greshake et al., 2023; Greshake, 2023). These studies, as well as subsequent ones (Yi Liu et al., 2023; Yupei Liu et al., 2023; X. Liu et al., 2024; Piet et al., 2023; Toyer et al., 2023; Yip et al., 2024), showed different categories of attacks, proposed various potential attacker objectives, and explored defensive ideas.

In May 2023, the Open Worldwide Application Security Project (OWASP) foundation announced that it would continue its popular Top 10 series to include one for LLMs (Wilson, 2023). Version 1.1 of the OWASP Top 10 for LLM Applications was published in October 2023 and includes the vulnerability types listed below (OWASP, 2023). For each vulnerability type, OWASP provides a comprehensive description, common examples, prevention/mitigation strategies, and example attack scenarios.

- LLM01: Prompt Injection
- LLM02: Insecure Output Handling
- LLM03: Training Data Poisoning
- LLM04: Model Denial of Service
- LLM05: Supply Chain Vulnerabilities
- LLM06: Sensitive Information Disclosure
- LLM07: Insecure Plugin Design
- LLM08: Excessive Agency
- LLM09: Overreliance
- LLM10: Model Theft

This paper explores continued ESCB development to integrate measures to protect against a subset of the vulnerability types enumerated by OWASP. We focus on LLM01 Prompt Injection, LLM04 Model Denial of Service, and LLM09 Overreliance. Prompt Injection (LLM01) protection efforts revolve around preventing prompts that return undesirable information. To mitigate the effects of Model Denial of Service (LLM04), we try to avoid allowing users to submit queries that might be overly resource-consuming. To avoid Overreliance (LLM09), we try to ensure the ESCB does not provide inaccurate information.

3. METHODOLOGY

In this paper, we again follow an action research approach to evolve the ESCB as we continue to address “questions in one’s immediate work environment, with the goal of solving an ongoing problem in that environment” (Leedy & Ormrod,

2010, p. 44). The canonical action research process model (Susman & Evered, 1978), Figure 3 (Davison et al., 2004), is followed to help ensure systematic rigor is applied to the problem. Steps include:

- Diagnosis – conduct a thorough examination of the current organizational circumstances
- Planning – the diagnosis results directly inform all planning; intended actions should be specified before being undertaken
- Action – planned actions are implemented in the order specified (if any)
- Evaluation – once planned actions are complete, outcomes are compared to project objectives and expectations
- Reflection – explicitly reflect on the activities taken and the outcomes achieved; decide whether to exit the cycle or iterate

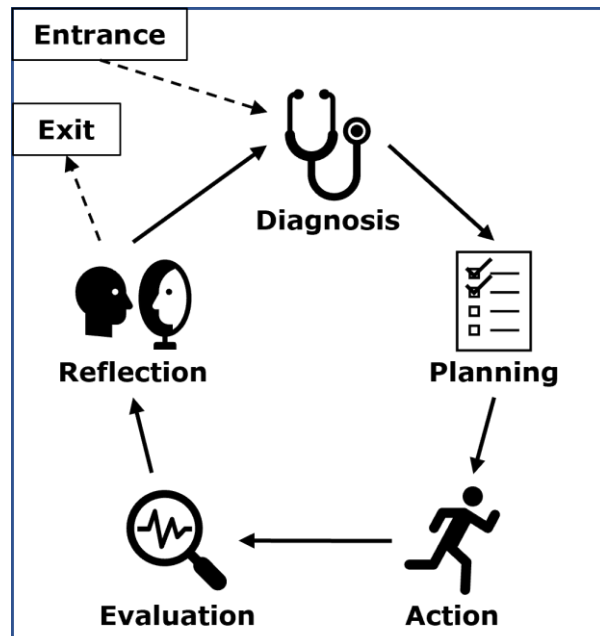


Figure 3: A canonical action research process model (Fig. 1. Davison et al., 2004)

Diagnosis and Planning

Our initial work on the ESCB successfully produced a chatbot capable of providing a dynamic customer service-oriented experience and of answering organization-specific questions. However, as we continued to use the system, it became apparent that the ESCB performance quality was achieved with well-behaved users and would not necessarily continue when used by those with ill intent. We also noticed that users’ varying query behavior might benefit from changes intended to improve performance. For example, frequently similar queries might provide an opportunity to use a cache to enhance performance. In contrast, the varied nature

(especially length) of other queries led us to suspect that the initial single LLM solution could be improved. It seemed apparent that with the diverse information requirements of an enterprise environment, the ESCB would benefit from a more nuanced approach to LLM employment, i.e., the ability to leverage more than one LLM.

To evolve the ESCB, we planned to continue to follow an iterative action research approach involving tool evaluation, coding, user interaction and feedback, and explicit results reflection. We iterated through two cycles to achieve the current state of the ESCB. Our aim remained to forge a replicable technique that other organizations could follow to develop and evolve their own ESCB. We modified the high-level conceptual sketch (Figure 1) and continued to refine it as we worked. The updated concept sketch in Figure 4 shows key components in rounded rectangles at the top, significant steps listed top-to-bottom, and arrows that indicate information flow between components. While this high-level sketch necessarily simplifies some of the more complex aspects of the process, it provides a clear overview of the ESCB's current operation.

From the diagnosis and planning steps conducted across two action research cycles, we envisioned the following improvements:

- Implement mechanisms to prevent the ESCB from returning undesirable information
- Make use of caching to return answers to recently asked similar queries quickly
- Analyze query complexity to guide LLM selection to improve ESCB performance

The following paragraphs briefly enumerate the actions taken to address the improvements. We will discuss some of the challenges and implementation details of these action steps in Section 4, Results & Discussion.

Action – LLM Firewall (FW)

Given prompt injection concerns, it seemed clear that user query input text should be processed more carefully before calling the GPT API. For this task, we created an LLM firewall (FW) that evaluated the query text and rejected it if it seemed likely to generate an undesirable response (more details in Section 4).

Action – Query Cache

Since users often have similar questions about organizational policy, some queries are very much like other queries. To take advantage of this fact for performance reasons, we implemented the well-known concept of caching so that answers to similarly phrased queries could be

returned immediately without calling an API (more details in Section 4).

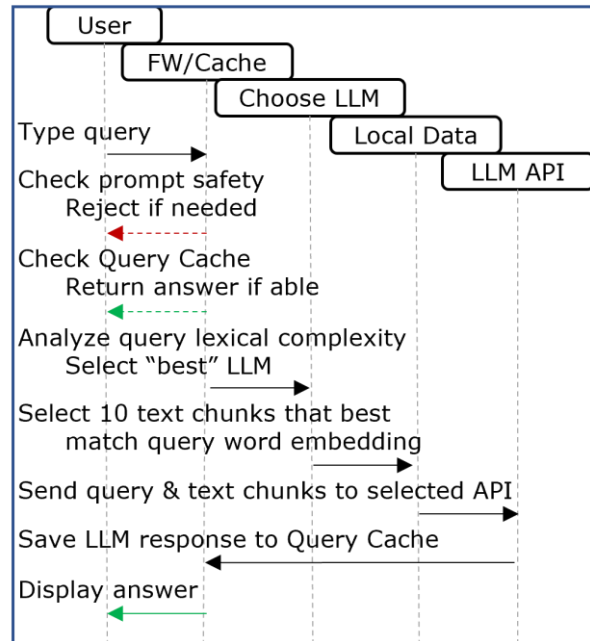


Figure 4: A logical depiction of significant steps in user-ESCB information exchange (top-to-bottom) and key components involved for the evolved ESCB

Action – Lexical Analysis & LLM Selection

For the initial ESCB, we looked at one LLM provider, evaluated the model offerings, and chose a single LLM API to service all queries. Given the availability of different LLM providers, each with multiple models of varied pricing and capability, we decided to explore the potential benefits of dynamically selecting which LLM API to call based on analysis of the user query. We limited the pool of possible APIs to three models, each from [OpenAI](#) and [Anthropic](#) (more details in Section 4).

Evaluation

To assess the functional performance of the evolved ESCB in order to guide development, we leveraged two opportunities to have people use the chatbot and provide feedback – similar to how we did during our initial three action research cycles. These settings allowed us to gather input from a range of users with varying levels of technical expertise.

For the first of the two latest cycles (fourth overall), we presented the evolved ESCB at a university research showcase, engaging with 20+ undergraduate and graduate students and professors from various disciplines. During the evaluation phase of the second (fifth) cycle, we

presented to eight software consultants. This evaluation offered insights from industry professionals and allowed for more rigorous testing of the system's capabilities in a more applied context. Across all five evaluation opportunities, more than 75 individuals, ranging from students and academics to IT professionals and industry consultants, used the ESCB. The progression of these evaluation cycles enabled us to make improvements that addressed challenges identified in earlier iterations.

The overwhelming majority of user feedback was supportive, quite possibly, in part, because of the amiable nature of the participants in the collegial venues at which we presented. As noted in our initial work, we quickly discerned that the clarity of the submitted query had a conspicuous effect on the quality of the ESCB reply, and users learned to be more specific in follow-up and subsequent queries. While many participants used the ESCB in a casual manner and were satisfied that it could answer basic policy queries, a few users were willing to spend a little more time probing the ESCB's capabilities. None of the feedback from these few was negative but rather often helped us see where the ESCB had room to improve. For example, one student wanted to know if he could vape inside campus buildings. Since the CLPD did not specifically address vaping, the ESCB could not directly answer the question and instead provided a generic and largely unhelpful response related to drug use policies.

Reflection

While the reflection phase of action research is enumerated last, it is an ongoing process integral to each cycle. Throughout this extended applied research activity, we employed deliberate reflection to assess ESCB development and to determine the need for additional action research cycles. The reflection activities helped us recognize that the ESCB improvements we made during the most recent two action research cycles, while non-trivial, were essentially proofs of concept and that additional cycles would be needed if more robust behavior was required.

Apparatus

Development and testing of the ESCB were conducted on a high-performance workstation configured for lightweight AI and machine learning tasks with the following specifications:

- Central Processing Unit (CPU): Intel Core i9-14900K, 3.20 GHz base clock speed
- Memory: 64 Gigabytes of DDR5 RAM
- Graphics Processing Unit (GPU): NVIDIA GeForce RTX 4090, 24 GB GDDR6X memory

- Storage: 2 TB NVMe SSD

We continued using the original 194-PDF-document CLPD (AKA Local Data) from our initial work as we focused on ESCB enhancements rather than expanding or purifying the knowledge base.

4. RESULTS AND DISCUSSION

This section examines the key components implemented during ESCB evolution, how they enhance security and effectiveness, and notes some limitations. We present some of the implementation details of the LLM FW, the Query Cache, and LLM model selection, which includes lexical analysis, [LangChain frameworks](#), and OpenAI word embeddings. Each element represents an important step in our iterative development process, addressing specific challenges and advancing our understanding of ESCB design and implementation. Figure 5 identifies how these key elements relate to each action research cycle iteration.

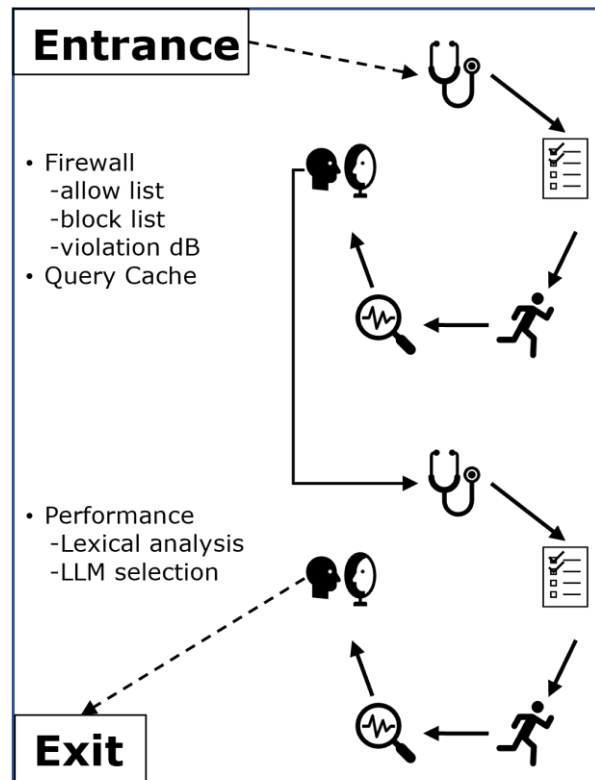


Figure 5: Highlights from the two additional action research cycles for ESCB evolution.

LLM FW Implementation

Developing an LLM FW was a critical component of the ESCB research, aimed at creating

lightweight middleware to intercept and filter malicious queries. We implemented the LLM FW via a block list, a violation list, and an LLM prompt safety check. An allow list was also conceptualized to maintain the ESCB's focus on its intended domain, though this functionality was primarily implemented as a proof of concept.

Two primary approaches to processing query language mirror how a traditional network FW handles network traffic, i.e., deny-by-default vs. allow-by-default. Established best practice for a network FW is deny-by-default, where traffic that is not trusted and explicitly allowed by exception does not pass the FW. We attempted to follow this best practice and created an *allowlist.json* file containing approved words and phrases against which user query language could be compared. Testing quickly revealed that generating an appropriately robust list that would allow almost all legitimate user queries was more difficult than anticipated. So, we left the code that could implement the allow list in place but disabled it for this version of the ESCB.

We next implemented a query check against a *blocklist.json* file containing words and phrases deemed inappropriate or out of organizational scope for the ESCB. We recognize the potential fragility of this approach as it might fail to block queries with ill intent that do not use disallowed words or phrases, while it might also block innocent queries that happen to contain a disallowed word or phrase. Despite this weakness, the testing and evaluation results demonstrated promise as the LLM FW watched for phrases such as 'bypass security' and 'ignore guidelines' and specific words related to violence or illegal activities. While simplistic, this approach effectively flagged queries containing these exact patterns.

To augment the block list, we created a *violation_calls.json* file that contains examples of previous prompt injections. To potentially catch cleverly worded user queries trying to circumvent the block list, we checked the cosine similarity of queries against the previously attempted exploits in this file. If the value exceeded a specified value (heuristically set to .65 after testing), the query was rejected and added to the JSON file. In this way, the LLM FW could "learn" about new prompt injections that should be blocked.

As a final layer of defense, we implemented a secondary prompt that operates in the background, unseen by the ESCB user. This prompt asks the LLM to evaluate whether the user's input is safe and aligns with the ESCB's

intended use. The prompt structure is depicted in Figure 6 and shows how the user's query input is included in the prompt.

This method proved very effective at catching sophisticated attempts to bypass the block list and violation database. Testing revealed several instances where this safety check successfully intercepted potentially harmful queries that had evaded the other two layers of protection.

```
"A user just sent this in, and we have been hacked before, is the following prompt allowed, meaning it is a valid question with no malintent (true), or does it intend harm or modify the large language model (LLM) to do things or roleplay how it should not normally (false)? Note legitimate users typically are using this chatbot to ask about policy documents but any question is fine so long as they are not attempting to circumvent the acceptable use. Do not explain, only respond with 'true' or 'false': '{user_input}'"
```

Figure 6: Secondary safety prompt

Query Cache Implementation

This feature stores all user queries, enabling immediate responses to previously asked questions, thus reducing redundant API calls and improving response times. Implementing an ESCB query cache was a response to observing the repetition of similar queries. We developed a no-SQL database structured in JSON format, stored locally on the ESCB's host system. We stored historical chat data in key-value pairs, including the original question, the provided answer, and a word embedding of the question generated using the Sentence Transformers Python library and the lightweight language model *paraphrase-MiniLM-L6-v2*.

When a query gets past the LLM FW, the system checks the cache, and if a similar query is found, the stored answer is returned immediately, bypassing the need for an LLM API call. This approach reduces latency, improves cost efficiency by reducing the number of API calls to external LLM services, and provides consistent answers for similar queries.

Implementing a query cache presents challenges. As the cache grows, adequate storage and retrieval mechanisms become crucial to maintain performance. Looking forward, we see potential for further improvements here. These could include implementing a time-based expiration for stored answers to ensure information freshness, developing more sophisticated similarity-matching algorithms to identify semantically

equivalent questions with different phrasings, and exploring distributed caching solutions for scalability in enterprise environments.

Dynamic LLM Selection

During evaluation, it seemed apparent that not all queries required calls to the most advanced (and expensive) LLM API. Since LLM choice can impact ESCB performance, cost-efficiency, and capability, we researched various LLM providers and models. We narrowed our focus to two AI leaders and their models: OpenAI's GPT-3.5-turbo, GPT-4-turbo, and GPT-4, and Anthropic's, Claude-3-haiku-20240307, Claude-3-sonnet-20240229, and Claude-3-opus-20240229.

We tested these models and noted differences in their output quality and appropriateness for various queries, confirming our intuition that the ESCB could benefit from dynamic LLM selection. Our analysis primarily focused on the relationship between model size, as indicated by parameter count and computational requirements, and the quality and relevance of responses. We observed that larger models generally produced more nuanced and contextually appropriate responses, especially for complex queries. However, this performance involved increased latency and higher API call costs. Interestingly, we found that the difference in response quality among models was less pronounced for simpler, more straightforward queries. Although we could not conduct an exhaustive comparison between the models, we noted that OpenAI models seemed to excel in general knowledge tasks, while Anthropic models showed strength in maintaining context over longer conversations.

For the ESCB to dynamically select an LLM, we determined to evaluate the lexical complexity of user queries and incorporate it into the selection apparatus. Based on the lexical complexity score, the ESCB automatically selects the most [presumably] suitable LLM for each query.

Lexical Complexity Analysis

Drawing inspiration from readability metrics used in linguistics, we calculated the lexical complexity for each query using the formula in Figure 7 that uses the following five components:

- *Readability Score*: We employ the Flesch Reading Ease score, implemented using the textstat Python library, to assess the overall readability of the query.
- *Lexical Richness*: Calculated as a type-token ratio, this measure reflects the diversity of vocabulary used in the query.
- *Semantic Diversity*: This is computed as the average cosine similarity between the vectors of all unique word pairs in the text, providing

- insight into the semantic range of the query.
- *spaCy Vector Norm*: We use the average word vector for the text, leveraging the spaCy library's pre-trained word embeddings.
- *Contextual Embedding Norm*: This is derived using BERT embeddings, with the mean value serving as the norm. This component captures deeper contextual nuances.

$$\text{complexity_score} = (1 - \text{read_score} / 100) + \text{lex_richness} + (1 - \text{sem_diversity}) + (\text{vector_norm} + \text{context_vector_norm}) / 100$$

Figure 7: Lexical complexity formula

We found that this method of lexical analysis offers several advantages:

- It provides a more objective basis for LLM selection than simple keyword or length-based approaches.
- The multi-faceted score helps account for different types of complexity (e.g., vocabulary richness vs. semantic depth).
- It allows for fine-tuned thresholds that can be adjusted based on the specific needs and resources of different ESCB implementations.

We also recognized some limitations in this approach. For instance, short queries with insufficient content can lead to unreliable scores. Additionally, some queries using simple language may still require complex reasoning, which our current lexical analysis might not fully capture.

A notable limitation in our lexical analysis approach occurs when dealing with multi-step reasoning questions. Queries using simple words but requiring complex, multi-step reasoning to answer adequately often resulted in selecting a less capable LLM. However, this limitation extends beyond our specific implementation – LLMs have historically struggled with multi-step reasoning problems. While this is largely outside the scope of expected use for an ESCB, it remains an important consideration.

LangChain Framework

The rapid evolution of AI technologies presented a challenge in maintaining the ESCB's relevance and functionality. Within a year of initial development, we found that OpenAI was discontinuing the API structure and endpoint we had used. This situation underscored the need for an adaptable and future-proof approach to ESCB development. We turned to open-source frameworks, specifically LangChain, for its extensive libraries and readily available pre-built components. LangChain offers the advantages of flexibility, standardization, and community

support that align with our research goals. For ESCB implementation, we specifically utilized the following LangChain components:

- **ChatAnthropic:** allows seamless integration with Anthropic's Claude models, enabling us to leverage their unique capabilities.
- **ChatOpenAI:** facilitates interaction with OpenAI's GPT models, maintaining our ability to use these widely adopted LLMs.
- **StrOutputParser:** helps process and standardize the output from different LLMs, ensuring consistency in ESCB responses.
- **ChatPromptTemplate:** allows for dynamic prompt construction, which is crucial for our adaptive query handling approach.

The adoption of LangChain significantly streamlined our development process. It allowed us to structure and format API calls consistently across different LLMs, facilitating easier comparison and integration of various models. Moreover, the framework's extensive documentation and examples provided a solid foundation for future experimentation and expansion of the ESCB's capabilities.

OpenAI Text Chunk Word Embeddings

We transitioned to using LangChain's integration with OpenAI's text chunking and embedding services. This shift was motivated by the need for improved efficiency and simplification of our codebase. We now utilize LangChain's document loader, text splitter, and vector store components in conjunction with OpenAI's embedding API. This approach allows us to maintain control over chunk size and number while leveraging the power of OpenAI's state-of-the-art embedding model. The new method offers improved embedding quality through continuously updated OpenAI models, reduced local computation requirements, a streamlined codebase, and easier maintenance.

***Code details available upon request**

5. FUTURE WORK

The action research cycles covered in this paper have illuminated several avenues for enhancing the ESCB. These potential improvements span security, efficiency, and scalability domains, each offering opportunities to refine and expand the capabilities of the ESCB.

Our experience with the LLM FW suggests that a more nuanced approach to detecting and mitigating potential misuse could be beneficial. This could involve developing more sophisticated algorithms for identifying emerging patterns of

malicious queries, moving beyond simple keyword blocking to a more context-aware security model. Further refinement of system prompts through advanced prompt engineering techniques could help address a broader range of edge cases, bolstering the ESCB's resilience against evolving prompt injection attacks.

Regarding efficiency, the current implementation of the Retrieval-Augmented Generation (RAG) process, while functional, leaves room for improvement. Future iterations could focus on streamlining the information retrieval process, potentially through more aggressive pre-processing and filtering of local documents to ensure that only the most relevant information is included in the ESCB's knowledge base. Furthermore, implementing a system of preemptive semantic sorting for document chunks could enhance the speed and accuracy of information retrieval during chat sessions.

Scalability represents a key consideration for the practical deployment of ESCBs in enterprise environments. Our research suggests that transitioning to a cloud-based infrastructure could offer significant advantages. By leveraging cloud services such as AWS, Azure, or Google Cloud, we could overcome local hardware limitations and facilitate easier scaling of the ESCB system. A particularly promising direction is serverless architecture. For instance, using AWS services like Lambda for chatbot logic and DynamoDB for data storage could enable a more flexible and scalable system that can dynamically adjust to varying usage demands.

However, transitioning to a serverless model would require careful testing and optimization. Notably, we would need to ensure that the lightweight machine learning models integral to our system, such as those used for semantic similarity calculations, can operate efficiently within the constraints of serverless environments. This might involve re-engineering specific components of the system or exploring alternative, cloud-optimized implementations of key algorithms.

These potential improvements, identified through the action research process, offer a roadmap for the continued evolution of our ESCB system. We aim to develop a more robust, adaptable, and enterprise-ready chatbot solution by addressing these security, efficiency, and scalability aspects. Future efforts will focus on implementing and evaluating these enhancements, further refining our understanding of effective ESCB design and deployment in real-world enterprise contexts.

An additional avenue for improvement centers on consolidating language models within our ESCB system. The current implementation utilizes various models for tasks such as embedding generation, query analysis, and response generation. While this approach has allowed us to leverage the specific strengths of different models, it has also increased the complexity of our codebase and potentially introduced inefficiencies in processing time. Transitioning to a single, versatile, lightweight LLM capable of handling all these tasks could streamline our system architecture and enhance overall performance. This consolidation would simplify our code and potentially decrease latency by eliminating inter-model switching.

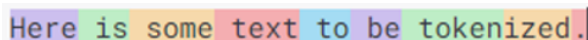
6. CONCLUSION

This paper presented the results of our continued use of applied research cycles to evolve the enterprise-specific chatbot (ESCB) project we introduced in earlier work (Wood & Stoker, 2024). Our research, now spanning five action research cycles after the two additional cycles described in this paper, demonstrated the potential of an ESCB and explored some of the challenges involved in its development.

Key aspects of this paper include the effectiveness of a multi-layered security approach, the benefits of dynamic LLM selection based on query complexity, and the advantages of leveraging open-source frameworks like LangChain for adaptability. The ESCB's evolution showcased improved capabilities in query filtering, response relevance, and operational efficiency. This research has significant implications for enterprise AI integration, highlighting the importance of balancing security, performance, and cost-effectiveness in chatbot implementations. As AI technologies advance, the insights gained from this study provide a foundation for developing more robust, efficient, and adaptable enterprise-specific AI solutions.

7. END NOTES

[1] tokenizes – breaks the query text into component words or word parts (Figure 8)



Here is some text to be tokenized.

Figure 8: 34 chars. broken into 9 tokens

[2] word embeddings – a natural language processing (NLP) technique that represents words as numbers in a way that preserves semantics

[3] centroid vector – a single vector representing an arithmetic mean, calculated by combining the vectors of each token created from the tokenized query or from the tokenized chunks of the CLPD.

[4] cosine similarity – ranging from -1 to 1, it is a metric that determines how alike two vectors (calculated from words) are

8. REFERENCES

- Ayanouz, S., Abdelhakim, B. A., & Benhmed, M. (2020, March). A smart chatbot architecture based NLP and machine learning for health care assistance. In Proceedings of the 3rd international conference on networking, information systems & security (pp. 1-6). <https://dl.acm.org/doi/10.1145/3386723.3387897>
- Branch, H. J., Cefalu, J. R., McHugh, J., Hujer, L., Bahl, A., Iglesias, D. d. C., Heichman, R., & Darwishi, R. (2022, September 5). Evaluating the Susceptibility of Pre-Trained Language Models via Handcrafted Adversarial Examples. <https://doi.org/10.48550/arXiv.2209.02128>
- Davison, R., Martinsons, M. G., & Kock, N. (2004). Principles of canonical action research. *Information Systems Journal*, 14(1), 65-86. <https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1365-2575.2004.00162.x>
- Goodside, R. [@goodside]. (2022, September 11). Exploiting GPT-3 prompts with malicious inputs that order the model to ignore its previous directions. <https://x.com/goodside/status/1569128808308957185>
- Greshake, K., Abdelnabi, S., Mishra, S., Endres, C., Holz, T., & Fritz, M. (2023, February 23). Not what you've signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection. <https://doi.org/10.48550/arXiv.2302.12173>
- Greshake, K. (2023, April 27). How We Broke LLMs: Indirect Prompt Injection. <https://kai-greshake.de/posts/llm-malware/>
- Leedy, P. D., & Ormrod, J. E. (2010). *Practical research, planning and design*, 9th edn, New Jersey: Pearson. <https://josemartimast.net/wp-content/uploads/2021/07/AP-Capstone-Research-Planning-and-Designing-E-Book.pdf>
- Liu, Yi, Deng, G., Li, Y., Wang, K., Wang, Z., Wang, X., Zhang, T., Liu, Y., Wang, H.,

- Zheng, Y., & Liu, Y. (2023, June 8). Prompt Injection Attack against LLM-integrated Applications. <https://doi.org/10.48550/arXiv.2306.05499>
- Liu, Yupei, Jia, Y., Geng, R., Jia, J., & Gong, N. (2023, October 19). Formalizing and Benchmarking Prompt Injection Attacks and Defenses. <https://doi.org/10.48550/arXiv.2310.12815>
- Liu, X., Yu, Z., Zhang, Y., Zhang, N., & Xiao, C. (2024, March 7). Automatic and Universal Prompt Injection Attacks against Large Language Models. <https://doi.org/10.48550/arXiv.2403.04957>
- Nuruzzaman, M., & Hussain, O. K. (2018, October). A survey on chatbot implementation in customer service industry through deep neural networks. In 2018 IEEE 15th International Conference on e-Business Engineering (ICEBE) (pp. 54-61). IEEE. <https://doi.org/10.1109/ICEBE.2018.00019>
- OpenAI. (2022, November 30). Introducing ChatGPT. <https://openai.com/index/chatgpt/>
- OWASP. (2023, October 16). OWASP Top 10 for LLM Applications, Version 1.1. https://owasp.org/www-project-top-10-for-large-language-model-applications/assets/PDF/OWASP-Top-10-for-LLMs-2023-v1_1.pdf
- Perez, F., & Ribeiro, I. (2022, November 17). Ignore previous prompt: Attack techniques for language models. <https://doi.org/10.48550/arXiv.2211.09527>
- Piet, J., Alrashed, M., Sitawarin, C., Chen, S., Wei, Z., Sun, E., Alomair, B., & Wagner, D. (2023, December 29). Jatmo: Prompt Injection Defense by Task-Specific Finetuning. <https://doi.org/10.48550/arXiv.2312.17673>
- preamble. (2022, September 22). Declassifying the Responsible Disclosure of the Prompt Injection Attack Vulnerability of GPT-3. https://www.preamble.com/prompt-injection-a-critical-vulnerability-in-the-gpt-3-transformer-and-how-we-can-begin-to-solve-it?trk=article-ssr-frontend-pulse_little-text-block
- Prompt Engineering. (2024, June 14). In Wikipedia. https://en.wikipedia.org/wiki/Prompt_engineering#Prompt_injection
- Shen, X., Chen, Z., Backes, M., Shen, Y., & Zhang, Y. (2024, May 15). "Do Anything Now": Characterizing and Evaluating In-The-Wild Jailbreak Prompts on Large Language Models. <https://doi.org/10.48550/arXiv.2308.03825>
- Susman, G. & Evered, R. (1978). An Assessment of The Scientific Merits of Action Research. *Administrative Science Quarterly*, (23) 4, 582-603. <https://doi.org/10.2307/2392581>
- Toyer, S., Watkins, O., Mendes, E. A., Svegliato, J., Bailey, L., Wang, T., Ong, I., Elmaaroufi, K., Abbeel, P., Darrell, T., Ritter, A., & Russell, S. (2023). Tensor trust: Interpretable prompt injection attacks from an online game. <https://doi.org/10.48550/arXiv.2311.01011>
- Willison, S. (2022, September 12). Prompt injection attacks against GPT-3. <https://simonwillison.net/2022/Sep/12/prompt-injection/>
- Wilson, S. (2023, May 23). Announcing the OWASP Top 10 for Large Language Models (AI) Project. <https://www.linkedin.com/pulse/announcing-owasp-top-10-large-language-models-ai-project-steve-wilson/>
- Wood, Z. & Stoker, G., (2024). An Action Research Approach to Building an Enterprise-Specific Chatbot (ESCB). *Journal of Information Systems Applied Research* 17(2) pp 61-73. <https://doi.org/10.62273/RAON2946>
- Yip, D., Esmradi, A., & Chan, C. (2024, January 2). A Novel Evaluation Framework for Assessing Resilience Against Prompt Injection Attacks in Large Language Models. <https://doi.org/10.48550/arXiv.2401.00991>