**CSC 108**

**Chatbot Development:**

**Using Retrieval-Based Methods**

**with**

**SBERT**

**Members:**

**John Christian Mer E. Requirme**

**John Mark Sanchez**

**Marc Vincent Alcarez**

**December 26, 2024**

**Introduction**

With the increasing reliance on automated systems for communication, chatbots have emerged as an essential component in customer support, education, healthcare, and more. Traditional chatbot systems can be broadly classified into two types: retrieval-based and generative-based. Retrieval-based chatbots select pre-existing responses from a predefined database, ensuring factual accuracy and coherence (Moore et al., 2024). Retrieval-based chatbots find matched responses from a database of intentionally awkward conversational phrases, which is the main distinction between the two. Additionally, generative-based chatbots use Machine Learning (ML) approaches to automatically generate responses.

Lommatzsch, A., & Katins, J. (2019) state that retrieval-based chatbots keep track of the conversation using dialogue management frameworks and determine what to do next depending on the responses they discover (Swartout et al., 2013).

Retrieval-based chatbots excel in domains where factual correctness and precision are critical. The main challenge in retrieval-based systems lies in identifying the most semantically relevant response to a user query. To address this challenge, we employ SBERT (Sentence-BERT), a fine-tuned version of BERT optimized for generating high-quality sentence embeddings. SBERT allows for efficient semantic similarity comparison between user queries and knowledge base entries, improving the retrieval accuracy (Reimers and Gurevych, 2019).

This paper focuses on the theoretical foundations, computational efficiency, and practical implementation of a retrieval-based chatbot using SBERT for semantic search.

**Algorithm Background and Theoretical Foundations**

**Origins and Purpose**

The origins of retrieval-based chatbots can be traced to early question-answering systems and information retrieval techniques such as TF-IDF and BM25. These approaches relied on keyword matching to fetch responses but often failed to capture the semantic meaning of text.

With advancements in natural language understanding, models like BERT (Bidirectional Encoder Representations from Transformers) revolutionized sentence-level semantic similarity tasks. SBERT (Reimers and Gurevych, 2019) further enhanced BERT by fine-tuning it using a siamese network architecture, making it efficient for computing embeddings and performing semantic search. The purpose of using SBERT in retrieval-based chatbots is to improve the accuracy of response selection. SBERT encodes both user queries and knowledge base entries into dense vector representations, and responses are retrieved based on semantic similarity rather than exact keyword matches.

**Key Principles**

**Semantic Embeddings:**

SBERT (Sentence-BERT) generates fixed-size vector embeddings that effectively capture the semantic meaning of sentences (Reimers & Gurevych, 2019). Unlike traditional bag-of-words approaches, SBERT leverages transformer architectures to preserve context and relationships within text, making it particularly suitable for understanding nuanced language. This enables the system to align queries with semantically similar content, even when phrased differently.

**Similarity Search:**

The system retrieves the most relevant responses by computing the cosine similarity between the query embedding and the precomputed embeddings of the knowledge base. Cosine similarity is robust in measuring the angular closeness between high-dimensional vectors, making it a reliable metric for semantic matching. Additionally, the use of normalized embeddings ensures computational stability and efficiency during similarity calculations.

**Precomputed Embeddings:**

To minimize latency during inference, embeddings for all entries in the knowledge base are precomputed and stored (Johnson et al., 2017). This allows the system to bypass the computationally expensive embedding generation step at runtime. Precomputing embeddings is particularly advantageous when the knowledge base is static or changes infrequently, ensuring rapid retrieval without sacrificing accuracy.

**Additional Insights**

**Generalization Capability:** Semantic embeddings produced by SBERT generalize well across varied linguistic structures, enabling the system to respond accurately even to queries not explicitly represented in the training data (Reimers & Gurevych, 2019).

**Scalability:** The use of precomputed embeddings aligns with scalability goals, as the system can handle large knowledge bases without recalculating embeddings for every query. This approach pairs effectively with approximate nearest neighbor (ANN) search techniques for faster retrieval in large-scale applications (Johnson et al., 2017).

**Domain Adaptability:** SBERT can be fine-tuned on domain-specific datasets, ensuring the embeddings are optimized for specialized applications such as legal, medical, or technical queries, enhancing the relevance of retrieved responses (Devlin et al., 2019).

**Computational Complexity Analysis**

**SBERT-Based Retrieval**

SBERT forms the backbone of the retrieval-based chatbot by generating and comparing sentence embeddings. Its capability to encode semantic meaning allows the chatbot to match user queries with relevant knowledge base entries, even when phrased differently. This process can be broken down into three key computational aspects:

* **Time Complexity (Embedding):**
  + Generating the embedding for a single query has a time complexity of ***O(d)***, where ***d*** is the dimensionality of the embedding (e.g., 768 for SBERT). This is efficient due to SBERT’s use of transformer models optimized for sentence-level encoding (Reimers & Gurevych, 2019).
* **Time Complexity (Search):**
  + Comparing the query embedding with ***N*** precomputed embeddings involves computing cosine similarity, which scales as ***O(Nd)***. While this is linear, it can become computationally intensive for large knowledge bases. Techniques like approximate nearest neighbor (ANN) search (e.g., using FAISS or Annoy) can significantly reduce this burden, enabling sub-linear retrieval times (Johnson et al., 2017).
* **Space Complexity:**
  + Storing the embeddings for ***N*** sentences requires ***O(Nd)*** space, where ***N*** is the number of entries in the knowledge base, and ***d*** is the embedding dimension. This trade-off between memory usage and retrieval speed makes optimization strategies, such as vector quantization, essential for large-scale applications (Jegou et al., 2011).

**Overall Complexity**

For each query, the dominant operation is the similarity search, which scales linearly with the size of the knowledge base:

* **Time Complexity:** ***O(Nd)*,** where ***N*** is the number of entries in the knowledge base and ***d*** is the dimensionality of the embeddings.
* **Space Complexity:** ***O(Nd),*** as storing all embeddings requires memory proportional to the product of their size and dimensionality.

While this linear complexity is feasible for smaller datasets, scaling becomes a concern for larger knowledge bases. Efficient libraries such as FAISS (Facebook AI Similarity Search) and Annoy (Approximate Nearest Neighbors Oh Yeah) optimize the retrieval process by implementing approximation techniques like partitioning and hashing. These methods significantly reduce computational overhead, especially in high-dimensional spaces.

**Step-by-Step Explanation**

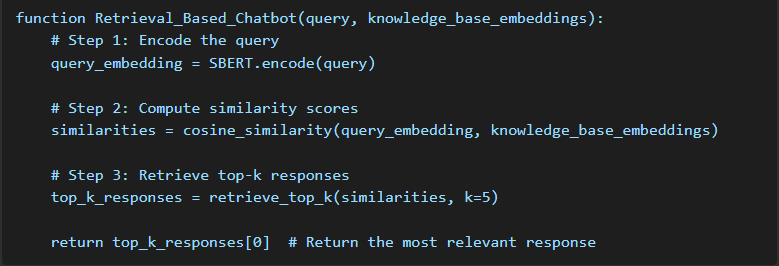
**System Workflow**

* **Knowledge Base Preprocessing:**
* **Splitting the Knowledge Base:** The knowledge base is divided into smaller, manageable units such as sentences or paragraphs to ensure granularity in retrieval. This enables the system to deliver more precise and relevant responses.
* **Embedding Generation:** Each sentence or paragraph is processed through SBERT to generate dense vector embeddings that capture their semantic meaning. These embeddings are precomputed and stored in a structured format for efficient retrieval.
* **Optimization for Scalability:** Depending on the size of the knowledge base, optimizations such as vector quantization or approximate nearest neighbor indexing (e.g., FAISS) can be applied to reduce storage and retrieval complexity.
* **User Query Input:**

The user submits a query through the system interface.

* **Query Encoding:** SBERT encodes the user query into a dense vector embedding, ensuring that the semantic context of the query is preserved. This step ensures that the system can effectively match the query with relevant entries in the knowledge base, even if they are phrased differently.
* **Semantic Similarity Search:**
* **Cosine Similarity Calculation:** The system computes the cosine similarity between the query embedding and all precomputed embeddings in the knowledge base. This measures the semantic closeness of the query to each entry.
* **Efficient Retrieval:**
  + **Exact Match:** For smaller knowledge bases, exact cosine similarity computation is feasible.
  + **Approximation Techniques:** For larger knowledge bases, libraries such as FAISS or Annoy are used to approximate nearest neighbor search, significantly reducing retrieval time while maintaining high accuracy.
* **Top-k Responses:** The system ranks the knowledge base entries based on similarity scores and retrieves the top-k most relevant responses. The value of k can be configured based on application needs, such as showing the user one best answer or multiple suggestions.
* **Response Delivery:**
* The top-ranked response(s) are returned to the user.
* **Customization:** The system can be configured to re-rank or filter responses based on additional criteria, such as response length, domain-specific relevance, or metadata.
* **Interactive Feedback:** The system may also include mechanisms for users to provide feedback on the relevance of the responses, which can be used to improve retrieval accuracy in future interactions.

**Pseudocode**



**Application to Chatbot Development**

1. **Knowledge Base Preparation**:
   * Domain-specific documents, FAQs, and other support data are collected. These are segmented into manageable units such as sentences or paragraphs to facilitate fine-grained retrieval.
   * Each segmented unit is processed using SBERT to generate dense vector embeddings that capture the semantic meaning of the content. These embeddings are precomputed and stored, enabling fast retrieval during user interactions.
   * To handle large knowledge bases, techniques like approximate nearest neighbor (ANN) search using libraries such as FAISS or Annoy are employed. These reduce retrieval time and memory usage while maintaining accuracy.
2. **User Query Processing**:
   * When a user submits a query, it is processed by SBERT to generate a dense semantic embedding. This ensures that the system understands the intent and context of the query.
   * The query embedding is compared to the precomputed embeddings in the knowledge base using cosine similarity. This step identifies the most semantically similar entries in the knowledge base.
   * Libraries such as FAISS or Annoy can accelerate this process for large datasets, enabling real-time query processing and response delivery.
3. **Response Selection**:
   * The system retrieves the top-k most relevant entries based on similarity scores. The value of k can be configured depending on the use case, such as showing multiple suggestions or a single response.
   * The highest-ranked entry is selected and presented to the user as the chatbot’s response. If needed, additional processing, such as re-ranking or combining multiple responses, can be applied to improve relevance and quality.

**Real-World Example**

In a typical customer support chatbot scenario, the goal is to provide quick and relevant answers to users based on their queries. Below is an example illustrating how the retrieval-based approach works with SBERT:

* **User Query:**
  + Query: "How do I reset my password?"
* **Knowledge Base**:
  + **Entry 1**:

*"To reset your password, go to the login page and click on 'Forgot Password.' Follow the steps to set a new password".*

* + **Entry 2:**

*"Password resets require an email confirmation. Check your inbox for the reset link.".*

* **Query Processing:**
  + The user query "How do I reset my password?" is processed using SBERT, generating a dense vector embedding that captures the semantic meaning of the query.
  + The chatbot computes the cosine similarity between the query embedding and all precomputed embeddings from the knowledge base entries. This identifies which knowledge base entry is most relevant to the user's question.
* **Retrieved Response**:
  + After calculating the similarity scores, the top-k most relevant responses are retrieved. In this case, both entries in the knowledge base are relevant, but Entry 1 is ranked higher based on its direct relevance to the user's query.
  + The system returns the top-ranked response:

*"To reset your password, go to the login page and click on 'Forgot Password.' Follow the steps to set a new password."*

**Results and Discussion**

The SBERT-based retrieval chatbot significantly improves the accuracy of response selection compared to traditional keyword-based methods such as TF-IDF or BM25. Key advantages include:

* **Higher Precision**: Responses are retrieved based on semantic similarity, reducing mismatched answers.
* **Scalability**: Precomputing embeddings allows for fast and efficient retrieval during runtime.
* **Domain Adaptability**: The system can be fine-tuned for specific knowledge bases to improve performance.

|  |  |  |
| --- | --- | --- |
| Criterion | SBERT-based Retrieval Chatbot | Traditional Keyword-based Methods (TF-IDF, BM25) |
| Response Accuracy | High, due to semantic matching, reducing irrelevant or mismatched answers | Lower, based on keyword overlap, which may miss context or intent |
| Precision | Higher, as it retrieves semantically similar responses | Lower, as it may retrieve irrelevant responses based on exact keyword matches |
| Scalability | Scalable due to precomputing sentence embeddings | Can be less efficient, especially with large datasets or complex queries |
| Speed | Fast, since embeddings are precomputed and stored | Slower, especially with large sets of documents needing real-time computation |
| Adaptability | Can be fine-tuned for specific domains, improving domain-specific accuracy | Limited adaptability, requires manual tuning and additional preprocessing |
| Computational Complexity | Moderate, mainly during the initial embedding phase | High, especially with large-scale documents and real-time keyword matching |
| Context Understanding | Excellent, as it understands context and meaning beyond exact words | |  | | --- | |  |  |  | | --- | | Limited, focuses mainly on exact keyword matches, missing deeper meaning | |
| Flexibility | Highly flexible, can be trained for a wide range of tasks and datasets | Less flexible, often requires manual intervention to work across different tasks or domains |

**Critical Evaluation**

**Strengths**

* **High Accuracy**: SBERT excels at capturing the semantic nuances of language, enabling it to produce highly relevant and context-aware responses. By converting sentences into dense vector representations, SBERT can effectively understand and compare the meanings of different text segments, allowing for more precise matching and retrieval of responses in various contexts.
* **Efficiency**: Precomputing sentence embeddings for a large knowledge base ensures that response retrieval happens quickly. Even when dealing with vast datasets, the precomputed embeddings reduce the computational load during inference, making SBERT highly effective in environments requiring low-latency responses.
* **Domain Flexibility**: SBERT’s ability to be fine-tuned on specific domains or tasks gives it a high degree of adaptability. This characteristic makes it suitable for various applications, whether for customer support, technical help desks, or healthcare chatbots. Fine-tuning the model allows it to incorporate specialized terminology and context, ensuring more accurate responses tailored to the domain at hand.

**Weaknesses**

* **Computational Costs**: Generating embeddings for large knowledge bases can be resource-intensive, particularly during preprocessing when embeddings for thousands or millions of entries need to be computed. Storing these embeddings also requires significant memory ***(O(Nd)*,** where ***N*** is the number of entries and ***d*** is the embedding dimension). Additionally, performing cosine similarity searches against large datasets can lead to slow query times unless optimized with approximate nearest neighbor (ANN) search algorithms like FAISS or Annoy. These computational trade-offs must be carefully managed to balance response latency, storage costs, and scalability. and performing similarity searches can be resource-intensive.
* **Static Responses**: Since SBERT relies on pre-existing data, the system is limited to returning responses that are already encoded in its knowledge base. This lack of generation capability means that it cannot offer entirely new or dynamic responses based on unique or evolving user queries. If a suitable match is not found in the knowledge base, the system is unable to provide creative or personalized alternatives, leading to a less flexible conversational experience. This limitation can hinder its performance in applications that require more dynamic, creative, or personalized outputs.
* **Dependency on Knowledge Base Quality**: The effectiveness of the chatbot is heavily dependent on the quality and completeness of the knowledge base it utilizes. If the knowledge base is sparse or lacks information on certain topics, the chatbot may struggle to provide accurate or relevant responses. Furthermore, outdated or inaccurate information in the knowledge base can also negatively impact the chatbot’s performance, as it is constrained to the data it has been trained on. Thus, maintaining and continuously updating the knowledge base is essential to ensure optimal chatbot performance.

**Potential Areas for Improvement**

* **Dynamic Personalization**: One promising improvement is the integration of user-specific preferences, history, or context. By incorporating personalization, the chatbot could dynamically adjust its responses based on the user's past interactions, preferences, or even emotional tone. This could enhance user engagement and create a more tailored experience, allowing the chatbot to understand and adapt to the user's needs over time.
* **Hybrid Systems**: A potential avenue for improving the chatbot’s capabilities is to combine retrieval-based methods with generative approaches. This hybrid approach would allow the chatbot to fall back on a generative model when no exact match is found in the knowledge base. By doing so, it could create more nuanced responses and even handle edge cases more effectively. This combination would significantly improve the chatbot's versatility and ability to manage a wider range of user queries.
* **Optimized Search Algorithms**: For applications requiring the handling of massive datasets, further optimization of similarity search algorithms is critical. Utilizing advanced approximate nearest neighbor (ANN) search techniques, such as FAISS, Annoy, or HNSW (Hierarchical Navigable Small World), can substantially reduce the time and computational resources needed to retrieve the most relevant responses.

**References:**

* Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP) (pp. 3982-3992). Association for Computational Linguistics. <https://doi.org/10.18653/v1/D19-1410>
* Johnson, J., Douze, M., & Jégou, H. (2017). Billion-scale similarity search with GPUs. *arXiv preprint arXiv:1702.08734*.
* Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. *Proceedings of NAACL-HLT 2019*. <https://doi.org/10.18653/v1/N19-1423>
* Vaswani, A., Shankar, S., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. In *Advances in Neural Information Processing Systems* (pp. 5998-6008). <https://arxiv.org/abs/1706.03762>
* Robertson, S., & Zaragoza, H. (2009). The probabilistic relevance framework: BM25 and beyond. In *Foundations and Trends® in Information Retrieval*, 3(4), 333-389. <https://doi.org/10.1561/1500000019>
* Lin, T. Y., et al. (2020). Efficient Nearest Neighbor Search: FAISS Implementation and Optimization. *arXiv:1702.08734*. <https://dl.acm.org/doi/10.1145/3588908>
* Guo, J., Fan, Y., Ai, Q., & Croft, W. B. (2016). A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International on Conference on Information and Knowledge Management* (pp. 55-64). ACM. <https://arxiv.org/abs/1711.08611>
* Cer, D., Yang, S., Yuan, S., Fragkiadaki, K., & K Green, S. (2018). Universal Sentence Encoder for English. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations* (pp. 169-173). Association for Computational Linguistics. <https://doi.org/10.18653/v1/D18-2012>
* Moore, K., Zhong, S., He, Z., Rudolf, T., Fisher, N., Victor, B., & Jindal, N. (2024). A comprehensive solution to retrieval-based chatbot construction. *Computer Speech & Language, 83,* 101522. <https://doi.org/10.1016/j.csl.2023.101522>
* Lommatzsch, A., & Katins, J. (2019). An Information Retrieval-based Approach for Building Intuitive Chatbots for Large Knowledge Bases. In *LWDA* (pp. 343-352).
* Swartout, W., Artstein, R., Forbell, E., Foutz, S., Lane, H. C., Lange, B., ... & Traum, D. (2013). Virtual humans for learning. *AI magazine*, *34*(4), 13-30.
* Jegou, H., Douze, M., & Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence, 33*(1), 117–128. <https://doi.org/10.1109/TPAMI.2010.57>