Building a Smarter Q&A System: What Worked, What Didn't

Christos Giannaros
Department of Mathematics/Department of Computer Science
University of Crete (UoC)

Abstract

This project takes a stab at building and evaluating a Question Answering (QA) system that digs out precise answers from a text collection. I tackled both factoid and confirmation questions by integrating various state-of-the-art (SOTA) Natural Language Processing (NLP) techniques. Critical components include a question and entity type classifier, a robust passage retrieval module, and an answer generation models. For factoid questions, the system leverages multiple BERT-based models (RoBERTa, DistilBERT, BERT-Large, BERT-Small) and a Large Language Model (LLM)(Llama-2), while a specialized cross-encoder handles confirmation Yes or No queries. To test if we can improve overall accuracy and robustness, the system integrates ensemble strategies, such ensemble of models by utilizing majority vote and semantic re-ranking, which combines model outputs of low confidence answers. Evaluation include Exact Match (EM), F1 score, and accuracy metrics, along with an analysis of computational latency, thus assessing the performance of individual components and the overall system. Results demonstrate that BERT-Large achieved the highest F1 score for factoid questions, while an 88.80% accuracy in passage retrieval significantly improved efficiency. Ensemble methods further enhanced overall answer reliability. The developed QA system presents a functional and adaptable framework for information extraction, highlighting the strengths of modern NLP models and strategic component integration in addressing diverse querying needs.

1 Introduction

This project details the development and evaluation of a comprehensive Question Answering (QA) system designed to extract answers directly from a given collection of texts. The system addresses both factoid and confirmation based questions, utilizing recent advancements in Natural Language Processing (NLP) and Large Language Models (LLMs) to provide accurate and efficient responses.

The primary objective is to build a robust QA framework that can accurately identify the correct context for a given question, classify the question type, and then generate precise answers. In this work we are going to evaluate the system performance using metrics such as Exact Match (EM) and F1 score, alongside execution times and classification accuracies for various techniques.

The methodology involves several stages. Initially, the system parses a curated collection of documents and their associated question-answer pairs. A baseline approach, based on random word selection, is established to provide a fundamental point of comparison for performance improvements. Then integrates a question classifier to predict whether a query is a factoid or a confirmation type, predicting the entity type for factoid questions.

For answering factoid questions, multiple SOTA transformer models, including various BERT architectures and the Llama-2 LLM, are employed and compared in terms of accuracy and latency. Confirmation questions are handled by a function that utilizes CrossEncoder models to determine "Yes" or "No" answers based on semantic similarity. To enhance overall accuracy, a passage retrieval mechanism is implemented to identify the most relevant document from the collection for any given query. Furthermore, various ensemble strategies are explored to combine the strengths of different models, aiming for superior performance.

This report presents the design choices, experimental setup, and quantitative results for each component of the QA system. It includes detailed analyses of model performances, discussions on the effectiveness of improvement strategies, limitations, and proposes future direction that this project can take to improve the question answering system performance.

2 Background and Related Work

The QA field has evolved significantly, moving from early rule based and information retrieval focused systems to neural network architectures. A QA system aims to automatically answer questions posed in natural language by retrieving information from a given text or knowledge base. This project focuses on document based QA, where the system processes a collection of documents to find precise answers.

QA systems typically handle different types of questions. Factoid questions seek specific pieces of information, such as names, dates, or locations, often requiring extractive answers directly from the text. Examples include "When was X born?" or "Where is Y located?". Confirmation questions, on the other hand, require a "Yes" or "No" answer, assessing whether a stated fact is true or false based on the provided context. Answering these often involves evaluating the semantic understanding of a text.

The foundation of modern NLP and QA systems is the Transformer architecture [1], introduced in 2017, transformers utilize self-attention mechanisms to weigh the importance of different words in a sentence, capturing long range dependencies and complex contextual relationships. This architecture forms the foundation for many pre-trained language models.

Among the most popularized transformer based models are the BERT (Bidirectional Encoder Representations from Transformers) models [2]. Pre-trained on vast amounts of text data using unsupervised learning tasks, enabling them to develop a deep understanding of language. For QA, BERT models are typically fine-tuned on labeled datasets like SQuAD [3], allowing them to identify answer spans within a given context. Variations such as RoBERTa, DistilBERT, and larger BERT models have demonstrated impressive performance in extractive QA tasks due to their robust contextual embeddings.

Recent advances brought us to Large Language Models (LLMs) like Llama-2 [4] have emerged as powerful tools for generative QA. These models are capable of not only extracting information but also synthesizing and generating human like text as answers. Their ability to reason and summarize

makes them capable for various QA scenarios, though their computational demands and potential for ”hallucinations”, generating plausible but incorrect information, require careful consideration.

Semantic similarity plays a crucial role across multiple components of a sophisticated QA system. It involves measuring the amount to which two pieces of text words, sentences, or documents convey similar meanings. Libraries like Sentence-Transformers [5] are crucial in generating dense vector representations (embeddings) of text, where semantically similar texts are mapped to nearby points in a high-dimensional space. These embeddings are then used to calculate cosine similarity, which quantifies their semantic relatedness. This technique is crucial for tasks such as:

- Passage Retrieval: Identifying the most relevant document or sentence from a large collection that is likely to contain the answer to a given question.
- Confirmation QA: Comparing the semantic content of a question with potential ”Yes” or ”No” implications derived from the text.
- Ensemble Methods: Re-ranking candidate answers based on their semantic similarity to the question, especially when primary models provide low confidence or conflicting answers.

Traditional Machine Learning (ML) classifiers remain valuable for specific tasks. For example classifying question types factoid vs confirmation or predicting entity types (e.g., PERSON, DATE) can be effectively performed using models like Random Forests [6] or Support Vector Machines (SVMs) [7], trained on features derived from question embeddings. These classifiers can provide valuable metadata that guides subsequent QA stages.

Finally, the evaluation of QA systems is paramount to assess their effectiveness objectively. Standard metrics like Exact Match (EM) and F1 score are widely used to compare predicted answers against the ground truth answers [3][8]. EM measures strict verbatim agreement, while F1 score is a more forgiving token based metric that accounts for partial overlaps, providing a balanced view of precision and recall. Additionally, analyzing latency is crucial for understanding the real world applicability of different models.

This project integrates these various SOTA and traditional techniques to build a multi-component QA system, aiming to achieve high accuracy and provide insightful analysis into the strengths and weaknesses of each approach.

3 Methodology and System Architecture

This section details the design and implementation methodology of the QA system, aiming for modularity, including several interconnected components, each addressing a specific requirement of the project. The Python code for this project is primarily implemented within a Google Colab notebook.

3.1 Data Collection and Preprocessing

The first step involved robustly loading the provided evaluation collection, `project_collection_2024_25.json`. This collection contains 40 topics, each with a context text and a list of associated question-answer pairs. To parse and store all necessary information from the JSON collection into structured data types for easy access and manipulation throughout the system. A custom data model was defined using dataclasses in python, specifically QA as pairs including question, answer, type, and entity and Topic for each document containing ID, title, text, and a list of QA objects. The load_collection function was developed to read the JSON file and write these dataclasses. This function was designed to be flexible, handling variations in context field names like `text` or `context`.

A snippet of the function demonstrating this parsing logic can be found in Appendix C (Section C.1).

3.2 Baseline System: Random Word Selection

To establish a foundational performance benchmark, a simple baseline QA system was implemented, in order to provide a basic answer generation strategy for comparison against more advanced models.

The random_word function was created to extract all words from a given text context using regular expressions and then return a single word chosen uniformly at random. The performance of this baseline was evaluated using the EM and F1 score metrics, as detailed in Appendix A.

A snippet of the function is available in Appendix C (Section C.2).

3.3 Question and Entity Type Classification

Before answering, it is beneficial to understand the nature of the question. This module classifies questions into 2 types factoid or confirmation question and for factoid questions to predict the specific entity type like PERSON, DATE, ORG etc. The QuestionClassifier class was implemented, in which question texts were transformed into numerical embeddings using a pre-trained Sentence-Transformer model, specifically `sentence-transformers/all-MiniLM-L6-v2` [5]. This model generates dense, fixed-size vectors that capture the semantic meaning of sentences. Then RandomForestClassifier [6] from `scikit-learn` was chosen for both question type prediction and entity type prediction. This choice aligns with the successful use of ensemble methods in previous assignment 2, as referenced in Appendix B, but could also use SVM as reported to have similar high performance. In turn, the classifier was trained on a portion of the QA items from the collection and evaluated on a separate test set using an 80/20% split with a fixed random seed for reproducibility, to avoid any data leakage. Accuracy was reported for both question type and entity type predictions.

Snippet of the QuestionClassifier class are showcased in Appendix C (Section C.3).

3.4 Passage Retrieval

For a QA system operating over multiple documents, identifying the most relevant document (passage) for a given question is a critical first step. To accurately detect the correct text from the 40 topics in the collection that can answer a given question, thus narrowing down the search space for the QA models. The PassageRetriever class was developed, which utilizes a CrosseEncoder model, specifically `cross-encoder/ms-marco-MiniLM-L6-v2` [9], as it was the most downloaded relevant model in huggingface but also after a couple of tests performed the best. So basically, for a given question and a list of topic texts, the best_topic method generates semantic similarity scores for each question-text pair. CrossEncoders are particularly effective for this task as they collectively encode both inputs to derive a relevance score. The topic with the highest relevance score is selected as the most likely context containing the answer. The accuracy of this retrieval step, whether the top ranked topic matches the actual topic from which the question was drawn, was evaluated.

A representation of the PassageRetriever and its best_topic method is provided in Appendix C (Section C.4).

3.5 Factoid Question Answering Models

This component is responsible for extracting the actual answer span from the provided context, either the full topic text or a retrieved passage. Several pre-trained BERT based models were employed to answer factoid questions.

The BERTpipeline class was created to streamline the use of multiple BERT models. It initializes a transformers.pipeline [10] for "question-answering" task for each specified model. The models selected for comparison were:

- `deepset/roberta-base-squad2`
- `distilbert-base-cased-distilled-squad`
- `mrm8488/bert-small-finetuned-squadv2`
- `deepset/roberta-large-squad2`

For each model, the answer method performs QA, records the latency, and extracts the predicted answer text and confidence score. These outputs are then used to calculate EM and F1 scores against the gold standard answers. The structure of the BERTpipeline class can be reviewed in Appendix C (Section C.5).

An LLM was integrated to explore its capabilities in extractive QA, in order to leverage the generative and reasoning capabilities of an LLM for answering factoid questions. The LlamaQA class, inheriting from a base LlamaModelBase for common setup was implemented, given as part of a tutorial. The `meta-llama/Llama-2-7b-chat-hf` model [11][4] was loaded using AutoModelForCausalLM and AutoTokenizer from the transformers library. To manage computational resources, BitsAndBytesConfig was applied for 4-bit quantization, and device_map="auto" ensured efficient GPU utilization. A Hugging Face token (HF_TOKEN) was used for authentication provided by the professors. A specific LLAMA_QA_PROMPT was designed to guide the LLM to answer questions succinctly and directly from the provided text, emulating the "shortest possible phrase" requirement. The generate_text method handles text generation, and the answer method extracts the final response by parsing the LLM output, looking for an "ANSWER:" tag. Similar to BERT models, LLAMA's answers were evaluated for EM and F1 scores, and its inference latency was measured.

Details of the LlamaQA implementation, including the prompt structure, are available in Appendix C (Section C.6).

3.6 Confirmation Question Answering

Confirmation questions require a "Yes" or "No" answer, which differs from the extractive nature of factoid QA, which are going to help us accurately determine whether a statement implied by the question and context if it is true or false, responding with "Yes" or "No". Thus, implementing The ConfirmationAnswerer class, in which a CrossEncoder model, specifically `cross-encoder/stsb-distilroberta-base` [5], was chosen. This model is well-suited for semantic similarity tasks, predicting a score indicating how well a question and context align. To provide the most relevant context to the cross-encoder, the full topic text was first tokenized into individual sentences using nltk.sent_tokenize. A separate SentenceTransformer model, `sentence-transformers/all-MiniLM-L6-v2`, was used to embed the question and each sentence. Cosine similarity was then used to find the sentence most semantically similar to the question, which became the refined context for the cross-encoder. A crucial step involved empirically determining an optimal confidence threshold for the cross-encoders score. The system iterates through a range of thresholds 0.1 to 0.95 to find the one that yields the highest accuracy on the confirmation questions in the dataset. This ensures that the "Yes"/"No" decision is made with the best possible balance. If the cross-encoders score for the question-context pair exceeds the optimal threshold, the answer is "Yes"; otherwise, it is "No". The accuracy of the confirmation model was reported based on how well its "Yes"/"No" predictions matched the gold standard answers.

The ConfirmationAnswerer class and its functionalities are detailed in Appendix C (Section C.7).

3.7 Ensemble and Improvement Strategies

To further improve the performance of answering factoid questions, two distinct ensemble strategies were devised and implemented. The purpose of this step was to combine the strengths of individual QA models (BERT and LLAMA) to improve overall answer accuracy and robustness. These strategies were integrated directly into the main evaluation loop.

- **Majority Vote Ensemble**: This simple yet effective strategy combines the answer from the BERT model that exhibited the highest confidence for a given question with the answer provided by the Llama model. The majority_vote utility function with a collections.Counter based approach, determined the final answer based on the most frequently occurring prediction among the candidates. If Llama did not produce a valid answer, the best BERT answer was used as a fallback .
- **Semantic Re-ranking Ensemble**: This more sophisticated approach acts as a fallback mechanism for low-confidence scenarios. If the highest confidence score from all BERT models for a given question fell below a predefined threshold set empirically to 0.35 and a re-ranking process was triggered. A set of unique candidate answers was formed from the best BERT answer and the Llama answer. These candidate answers were then re-ranked based on their semantic similarity to the original question using the SentenceTransformer (`sentence-transformers/all-MiniLM-L6-v2`). The candidate with the highest

semantic similarity was selected as the final answer. If the best BERT confidence was above the threshold, the best BERT answer was directly used, bypassing the re-ranking.

The performance EM and F1 score of both ensemble strategies was measured and compared against the individual models. Detailed snippets of the strategies above can be found in Appendix C (Section C.8)

3.8 Evaluation Metrics

The performance of all QA components and models was quantitatively assessed using standard evaluation metrics. These include Accuracy for classification tasks and Exact Match (EM) and F1 score for answer generation tasks. Latency (execution time) was also measured for each QA model. A detailed explanation of these metrics and their formulas is provided in Appendix A.

4 Experimental Setup and Results

This section details the experimental environment used for developing and evaluating the QA system, presents the results for each implemented component, and includes a discussion of the observed performance, alongside any limitations and challenges faced.

4.1 Experimental Environment

The entire QA system was developed and executed within a Google Colab environment. This provided access to NVIDIA L4 GPUs, significantly accelerating the inference and training processes for deep learning models, but most of all enabled it as it was a very heavy process most commercial GPUs nowadays couldn't handle it. The primary programming language used was python, with libraries including:

- `transformers` [10] for BERT-based models and LLM integration.
- `sentence-transformers` [5] for generating universal sentence embeddings.
- `scikit-learn` [12] for machine learning classifiers (Random Forest) and utility functions.
- `nltk` for text tokenization like sentence tokenization. [13]
- `pandas` and `matplotlib`/`seaborn` for data handling, analysis, and visualization.
- `bitsandbytes` for 4-bit quantization of the Llama-2 model, optimizing memory usage and speed. [14]

4.2 Baseline Performance

The random word selection baseline served as a fundamental point of comparison. As expected, its performance was minimal, highlighting the necessity for more sophisticated QA strategies.

Table 1: Overall Performance Metrics

Metric	Value	N_Questions
Random word baseline	0.00% EM / 1.11% F1	250
Question type prediction accuracy	76.00%	50
Factoid entity type prediction accuracy	55.26%	38
Passage retrieval accuracy	88.80%	250

Note: EM and F1 for baseline are for factoid questions. Classifier and retrieval accuracies are for their respective tasks.

The random word baseline achieved an F1 score of 1.11% and an EM score of 0.00%, confirming its role as a lower bound for expected performance. This low score underscores the complexity of natural language understanding and question answering.

4.3 Question and Entity Type Classification Performance

The QuestionClassifier was trained and evaluated to determine its efficacy in categorizing questions.

- **Question type prediction**: The classifier achieved an accuracy of **76.00%** for distinguishing between factoid and confirmation questions on the test set. This indicates a solid capability in identifying the fundamental intent of a query.
- **Factoid entity type prediction**: For factoid questions, the classifier achieved an accuracy of **55.26%** in predicting the specific entity type. While lower than question type accuracy, this still provides a valuable signal for subsequent QA processes, especially for models that can leverage entity information.

The performance of the classifier is crucial for the overall system, as accurate type prediction allows the system to route questions to the most appropriate answering module factoid QA models vs. confirmation model.

4.4 Passage Retrieval Accuracy

The PassageRetriever played a vital role in identifying the correct context for factoid questions from the collection of 40 topics. The passage retrieval was the MVP (Most Valuable Player) with an achieved an impressive accuracy of **88.80%**. This means that for nearly 9 out of 10 questions, the retriever successfully identified the exact document containing the correct answer. This high accuracy significantly reduces the search space for the subsequent QA models and improves efficiency by providing a focused context. This result validates the use of `cross-encoder/ms-marco-MiniLM-L6-v2` for relevance ranking in a multi-document QA setup. We also use other encoder like `cross-encoder/qnli-distilroberta-base` or other Roberta type, but accuracy dropped to a staggering 63.60%, which signifying the importance of the CrossEncoder selection.

4.5 Confirmation Question Answering Performance

The ConfirmationAnswerer module was specifically designed to handle "Yes/No" questions. An empirical analysis identified an optimal threshold of 0.65 for the `cross-encoder/stsb-distilroberta-base` model [15]. This threshold achieved the highest accuracy in classifying confirmation questions. With this optimal threshold, the confirmation module achieved an accuracy of **58.00%**. This indicates a moderate ability to correctly answer "Yes/No" questions. The performance is visualized in figure 1.

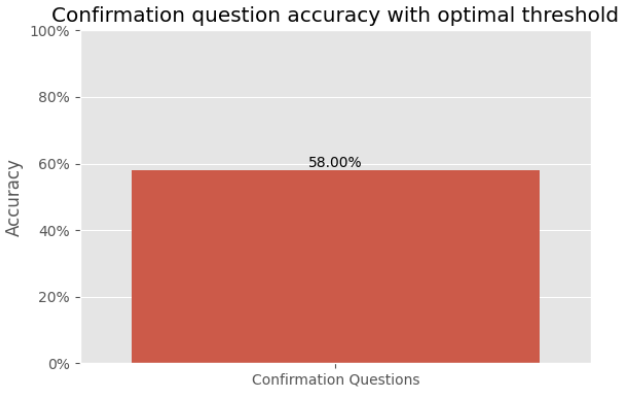

Figure 1: Confirmation Question Accuracy with Optimal Threshold

4.5.1 Limitations and Challenges in Confirmation QA

Despite the robust implementation using a cross-encoder and optimal thresholding, the confirmation QA accuracy of 58.00% suggests inherent challenges. Confirmation questions often involve subtle

negations or require inference beyond direct textual presence, which similarity based models can struggle with. For example, if the text states ”X happened in 1870” and the question is ”Did the X happen in 1834?”, the model needs to infer negation. Also, while sentence tokenization helps, a single ”best sentence” might not always capture all necessary nuances for a complex confirmation. And lastly, reducing complex textual entailment to a binary ”Yes/No” can be challenging when the text provides ambiguous or partial information.

4.6 Factoid QA Model Performance

The performance of individual BERT-based models and the Llama-2 LLM on factoid questions was a central focus.

Table 2: Per Model Performance for Factoid Questions

Model	EM	F1	Avg Latency (s)	N_Questions
bert-large	50.00%	65.97%	0.05544	200
roberta	49.50%	62.80%	0.02658	200
ensemble_rerank	48.50%	62.67%	0.00000 [1]	200
ensemble	47.00%	61.10%	0.00000 [1]	200
distilbert	43.00%	57.18%	0.01608	200
llama	33.50%	50.17%	1.27719	200
bert-small	37.00%	48.43%	0.01299	200

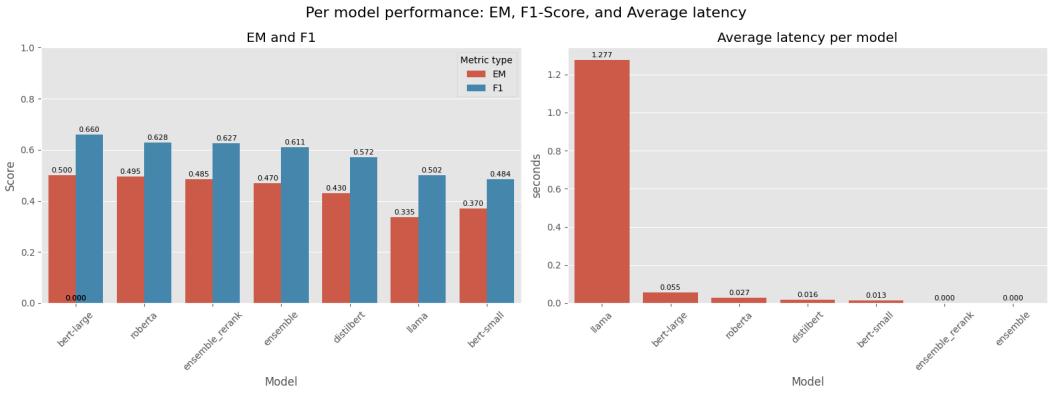

Figure 2: Per Model Performance: EM, F1 score, and Average Latency

4.6.1 Analysis of Factoid QA Model Performance

`Bert-large` consistently outperformed other BERT variants, achieving the highest F1 score of 65.97%. This is expected, as larger models generally capture more complex linguistic patterns. `Roberta` also showed strong performance with 62.80% F1 score, while `distilbert` with 57.18% F1 score and `bert-small` with 48.43% F1 score performed lower, which is typical due to their smaller size and faster inference times. As anticipated, smaller models like `bert-small` and `distilbert` offered the lowest latencies 0.013s and 0.016s respectively, making them suitable for applications requiring real-time responses. `Roberta` was also remarkably fast with 0.027s, while `bert-large` had a slightly higher but still very acceptable latency of 0.055s. Llama achieved an F1 score of 50.17%, placing it below most BERT models in terms of accuracy for this extractive QA task. Crucially, its average latency was significantly higher at 1.277s, approximately 20-100 times slower than the BERT models. This highlights a trade-off between model size/generative capability and inference speed.

[1]Latency for ensemble models is not independently measured as they combine results from other models whose latencies are already accounted for.

[1]

4.6.2 Limitations and Challenges in Factoid QA

While Llama-2 is a powerful generative model, its performance in this purely extractive task where a precise span is expected was not as strong as specialized extractive BERT models. This is partly due to the prompts challenge in forcing a short, precise extraction and Llama's tendency to paraphrase or add conversational filler. Running larger LLMs like Llama-2 requires significant computational resources, leading to higher latencies. Even with 4-bit quantization, the model remains slower than BERTs, limiting its real-time applicability without more powerful hardware or further optimization. The prompt engineering for LLMs to extract exact answers can be tricky. Llama might still generate slightly different wording than the gold standard, impacting EM and F1 scores even if the answer is semantically correct, this was also identified when tried newer Llama model 3.2 and 3.1 (1B-3B-8B params) instruct model as they performed much worst $<$0.05 F1 score on the same pipeline, this could be due the prompt used for retrieving the answer as modification and more complex prompt must take place, signifying that we cant use any model to our pipeline without necessary modification. Pre-trained models are general purpose, for highly specialized domains, fine-tuning on domain-specific QA datasets would likely yield better results. This project used a diverse, general collection, which might present varied challenges for models trained on general text.

4.7 Ensemble Model Performance

Two ensemble strategies were implemented to explore performance improvements by combining model strengths. The results for these ensembles are included in table 2 and figure 2.

- **Majority Vote Ensemble**: This basic ensemble combining best BERT and Llama achieved an F1 score of 61.10%. While this is better than individual Llama and smaller BERTs, it slightly trails the top performing `bert-large` and `roberta` models. This suggests that simple majority voting may not always capture the best answer if the top performing models disagree.
- **Semantic Re-ranking Ensemble**: This adaptive ensemble, which re-ranks candidates when BERT's confidence is low, achieved an F1 score of 62.67%. This performance is competitive with `roberta` and only slightly below `bert-large`, demonstrating its effectiveness as a robust strategy. It implies that leveraging semantic similarity for re-ranking in uncertain scenarios can effectively boost overall performance by mitigating the risks of low confidence predictions from primary models.

4.8 Entity-Level Performance

The performance analysis also included a breakdown of EM and F1 scores per entity type, providing insights into which types of information the best models handle more effectively. The metrics were computed using the answers from the highest-confidence BERT model for each question.

Table 3: Entity-Level Performance of F1 score of Best BERT per Question

Entity Type	EM	F1	N_Questions
NORP	100.00%	100.00%	1
NAME	100.00%	100.00%	1
PROFESSION	100.00%	100.00%	1
MONEY	100.00%	100.00%	1
PERCENT	100.00%	100.00%	2
TIME	50.00%	83.33%	2
DATE	65.38%	80.22%	26
LOC	60.00%	72.44%	15
EVENT	0.00%	71.15%	2
ORG	66.67%	66.67%	9
PERSON	51.52%	65.87%	33
CARDINAL	45.45%	54.39%	22
GPE	31.58%	51.57%	19
O	36.21%	50.37%	58
ASPECT	0.00%	50.00%	1
DATE, TIME	0.00%	42.86%	1
ORG, GPE	0.00%	40.00%	1
GENRE	0.00%	33.33%	1
FAC	33.33%	33.33%	3
QUANTITY	0.00%	0.00%	1

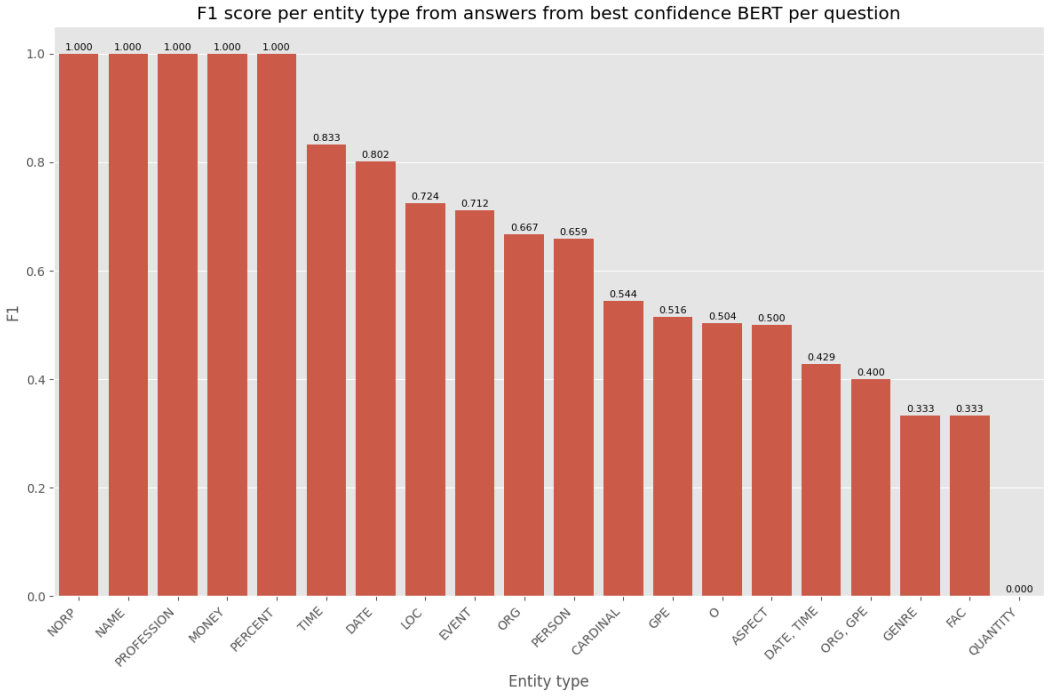

Figure 3: F1 score per entity type from answers from the best confidence BERT per question

4.8.1 Analysis of Entity-Level Performance

Entities like NORP, NAME, PROFESSION, MONEY, and PERCENT achieved 100% F1 scores. These are often distinct, well defined, and explicitly mentioned in the text, making them easier for models to extract accurately. DATE with 80.22% F1 score and TIME with 83.33% F1 score also performed very well, likely due to their structured format and clear presence in the text. LOC with

72.44% F1 score, EVENT with 71.15% F1 score, ORG with 66.67% F1 score, and PERSON with 65.87% F1 score showed good but not perfect performance. These entities can be more ambiguous, might appear in various forms, or require more contextual understanding for precise extraction. Entities like CARDINAL with 54.39% F1 score, GPE with 51.57% F1 score, and Other (O) with 50.37% F1 score proved more challenging. O often represents diverse, unstructured answers that are harder to generalize. Combinations like DATE, TIME with 42.86% F1 score or ORG, GPE with 40.00% F1 score are complex as they require extracting multiple distinct entities. Low or zero F1 scores for GENRE, FAC, and QUANTITY with 0% suggest that these types might be rare in the dataset low N_Questions or require specific reasoning, like calculations for QUANTITY that the models struggle with.

4.8.2 Limitations and Challenges in Entity-Level Extraction

Entities with very few occurrences like NORP, NAME, PROFESSION, MONEY, PERCENT, etc. might show deceptively high or low F1 scores due to small sample sizes. The low counts make it difficult to generalize the model's performance for these types. Some entity types, especially QUANTITY, might require numerical reasoning or aggregation like "How many rooms?" requiring summing up "4 superior double rooms and 2 double floor family apartments" that current extractive models are not inherently designed for. Discrepancies between how entities are annotated in the gold standard and how models interpret and extract them can impact scores.

5 Discussion and Analysis

This section provides a comprehensive analysis of the experimental results presented in section 4, interpreting the performance of each component and the overall synergy of the developed QA system. The aim is to highlight findings, observed trade-offs, and the effectiveness of the chosen methodologies.

5.1 Overall System Effectiveness and Component Synergy

The multi-component QA system demonstrated a significant leap in performance compared to the simple random word baseline in table 1, affirming the value of NLP techniques. The systems modular design allowed for specialized handling of different question types and contexts, contributing to its overall robustness.

The high accuracy of the Passage Retriever with 88.80% accuracy, table 1 stands out as a critical enabler. By effectively identifying the correct topic for a given question, it drastically reduced the search space for downstream QA models, ensuring they operated on the most relevant context. This preliminary filtering step is indispensable for scalability and efficiency in real-world multi-document QA scenarios.

The Question and Entity Type Classifier with 76.00% question type accuracy, 55.26% entity type accuracy, table 1 provided valuable metadata that could guide the QA process. While entity type prediction accuracy was moderate, it still offers a useful signal for systems that can leverage this information to constrain answer search or refine generative outputs. Its primary role in this system was to route confirmation questions to their specialized module, demonstrating effective component integration.

5.2 Factoid Question Answering: Performance, Trade-offs, and Ensemble Impact

The comparison of factoid QA models revealed distinct performance profiles and crucial trade-offs, as detailed in table 2 and figure 2. `bert-large` consistently emerged as the top performer among the BERT-based models for factoid QA, achieving the highest F1 score of 65.97%. This underscores the general principle that larger transformer models, with their increased capacity for learning intricate language patterns, tend to yield superior accuracy in complex NLP tasks like extractive QA. `roberta` also delivered very competitive results, confirming the effectiveness of RoBERTa's architecture. A clear trade-off was observed between model size and inference latency. Smaller models like `bert-small` and `distilbert` offered significantly faster response times of sub-0.02s latency run but at the cost of lower F1-scores. This makes them suitable for high-throughput, low

latency applications where some accuracy sacrifice is acceptable. `llama` showed a moderate F1 score with 50.17%, lower than most BERT models for this specific extractive task. Its most notable characteristic was its substantially higher latency of 1.277s. While LLMs possess immense generative power and broader reasoning capabilities, their direct application in a strictly extractive QA setting, especially with short, precise answers, can be less efficient than fine-tuned extractive models. The latency bottleneck highlights a challenge for deploying large LLMs in real time QA without specialized hardware or further distillation. The ensemble methods demonstrated their value in improving overall performance. The majority vote ensemble provided a slight improvement over individual smaller BERTs. More notably, the semantic re-ranking ensemble with 62.67% F1 score proved to be a robust strategy. By adaptively re-ranking candidate answers based on semantic similarity to the question when initial confidence was low, it successfully mitigated the risk of poor individual model predictions. This approach brings the overall F1 score close to that of the best individual BERT model, suggesting that intelligent combination strategies can indeed yield more reliable answers.

5.3 Confirmation Question Answering: Specificity and Thresholding

The accuracy of the ConfirmationAnswerer with 58.00% accuracy at an optimal threshold of 0.65, figure 1, indicates that confirmation questions present a unique set of challenges compared to factoid questions. While the cross-encoder is effective for semantic similarity, inferring a definitive "Yes" or "No" from nuanced textual information can be difficult. The empirical determination of an optimal threshold highlights the importance of calibrating the model's confidence for binary classification. This step is crucial for balancing precision, avoiding false positives, and recall, capturing all true confirmations.

5.4 Entity-Level Performance: Insights into Extraction Complexity

The entity-level F1 scores in table 3 and figure 3 offered granular insights into the types of information the systems best factoid QA model, primarily BERT-based, could extract most reliably. Categories like NORP, NAME, PROFESSION, MONEY, PERCENT, DATE, and TIME consistently achieved very high F1 scores. These entities often have clear, unambiguous textual representations and conform to recognizable patterns, making them easier for extractive models to pinpoint. Entities such as LOC, ORG, and PERSON, while generally well-handled, showed slightly lower scores. This can be attributed to their greater variability in expression, potential for ambiguity, or the need for more complex contextual understanding. Categories like O (Other), CARDINAL, EVENT, GENRE, FAC, and especially QUANTITY proved more challenging. O is inherently diverse and lacks a consistent pattern. QUANTITY questions, in particular, may often require arithmetic operations or inferential reasoning, like counting from context, rather than direct span extraction, which is a known limitation of extractive QA models. Similarly, EVENT or GENRE answers might be less explicitly defined spans, leading to lower F1 scores. The low number of questions for some of these challenging categories like QUANTITY, GENRE, ASPECT, also makes it difficult to draw robust conclusions about their overall performance.

6 Conclusion and Future Work

6.1 Conclusion

This project successfully designed, implemented, and evaluated a multi-component QA system capable of extracting answers from a given text collection. The system effectively addressed both factoid and confirmation questions by integrating various SOTA NLP and ML techniques. A reliable data loading mechanism was established, and a random word baseline provided a clear lower bound for performance comparison. The Passage Retriever demonstrated high accuracy of 88.80%, proving indispensable for narrowing down the context. The Question Classifier, while more challenging for entity types with accuracy at 55.26%, still provided valuable routing for question types at 76.00% accuracy. Specialized BERT models, particularly `bert-large`, excelled in extractive QA, achieving the highest F1 score of 65.97%. This confirmed the effectiveness of fine-tuned transformer architectures for precise span extraction. The Llama-2 LLM, despite its advanced capabilities, showed a lower F1 score of 50.17% and significantly higher latency of 1.277s compared

to the best BERT models for this specific extractive task, highlighting a trade-off between model generality/generative power and efficiency/precision in extractive QA. A dedicated confirmation answering module with an optimal threshold of 0.65 achieved 58.00% accuracy, demonstrating a viable approach for binary "Yes/No" questions. The implemented ensemble strategies, especially the semantic re-ranking approach, successfully improved overall F1 scores of 62.67%, indicating the potential of combining diverse models to enhance robustness and accuracy. Entity level performance analysis provided detailed insights into the systems strengths of high accuracy for DATE, PERSON, ORG and weaknesses like the challenges with QUANTITY or (O)ther entity types. Overall, the developed QA system serves as a solid framework for information extraction, demonstrating the power of current NLP models and strategic component integration.

6.2 Future Work

Building upon the current systems capabilities and addressing identified limitations, several avenues for future work can be explored to further enhance its performance and applicability, like integrating dedicated Named Entity Recognition (NER) and Entity Linking (NEL) tools like those explored in Assignment 1, as a pre-processing step for factoid questions. This could provide more precise entity spans or IDs to guide the QA models, potentially improving accuracy for challenging entity types like GPE, LOC, or ORG. Experiment with various prompt engineering techniques to better constrain LLMs to extractive answer formats, potentially using few-shot examples within the prompt itself. Investigate smaller, more specialized LLMs or explore techniques like knowledge distillation to reduce LLM latency while maintaining performance. Explore LLMs for their reasoning capabilities like for QUANTITY questions requiring calculation rather than just extraction. Implement a mechanism to detect when a question cannot be answered from the provided text collection like using a confidence threshold for rejection or by querying an external knowledge base like Wikipedia/DBpedia. By pursuing these directions, the QA system can evolve into an even more accurate, efficient, and versatile tool for extracting information from textual data.

References

[1] A. Vaswani et al., "Attention is all you need," *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4171–4186, 2019.

[3] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang, "Squad: 100,000+ questions for machine comprehension of text," *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pp. 1–10, 2016.

[4] H. Touvron et al., "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023.

[5] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3981–3990, 2019.

[6] L. Breiman, "Random forests," *Machine Learning*, vol. 45, no. 1, pp. 5–32, 2001.

[7] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, no. 3, pp. 273–297, 1995.

[8] S. Kierszbaum, *F1 score in nlp-span-based qa task*, `https://kierszbaumsamuel.medium.com/f1-score-in-nlp-span-based-qa-task-5b115a5e7d41`, Accessed: 2025-06-03, 2020.

[9] N. Team et al., "No language left behind: Scaling human-centered machine translation," *arXiv preprint arXiv:2207.04672*, 2022.

[10] T. Wolf et al., "Huggingface's transformers: State-of-the-art natural language processing," *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pp. 38–45, 2020.

[11] Meta and H. Face, *Llama 2 7b chat hf*, `https://huggingface.co/meta-llama/Llama-2-7b-chat-hf`, Accessed: January 8, 2026, 2023.

[12] F. Pedregosa et al., "Scikit-learn: Machine learning in python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

[13] S. Bird, E. Klein, and E. Loper, *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.

[14] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, "8-bit optimizers via block-wise quantization," *9th International Conference on Learning Representations, ICLR*, 2022.

[15] N. Reimers, *Cross-encoder/stsb-distilroberta-base*, `https://huggingface.co/cross-encoder/stsb-distilroberta-base`, Accessed: January 8, 2026, 2020.

A Appendix A: Evaluation Metrics Formulas

This appendix provides the detailed formulas for the evaluation metrics used throughout this report: Exact Match (EM) and F1 score. These metrics are standard in the field of QA for assessing the accuracy and quality of generated answers.

A.1 Accuracy

Accuracy is used to measure the performance of classification tasks, such as Question Type Prediction or Passage Retrieval. If Q denotes the set of all questions used in the evaluation, and AQC represents the subset of questions that were answered correctly by the system, then Accuracy is defined as the fraction of correctly answered questions:

$$\text{Accuracy} = \frac{|AQC|}{|Q|}$$

The number of wrongly answered questions is also referred to as error, where Error $= 1 -$ Accuracy.

A.2 Exact Match (EM)

Exact Match is a strict metric that assesses whether a predicted answer precisely matches the ground-truth answer. For each question-answer pair $(prediction, reference)$, the EM score is:

$$\text{EM} = \begin{cases} 1 & \text{if prediction}_{\text{normalized}} = \text{reference}_{\text{normalized}} \\ 0 & \text{otherwise} \end{cases}$$

Where $\text{prediction}_{\text{normalized}}$ and $\text{reference}_{\text{normalized}}$ refer to the answers after normalization, lowercasing and stripping whitespace. This is a metric, which even a single character difference results in a score of 0. For negative examples like confirmation questions where the answer is 'No' and the model predicts any text, an EM of 0 is automatically assigned.

A.3 F1 score

The F1 score is a widely used metric in QA that provides a balanced measure of precision and recall, especially relevant for tasks where answers might not be exact matches but have significant token overlap. It is computed at the token level by tokenizing both the predicted answer and the ground truth answer.

Precision is the ratio of the number of shared tokens between the prediction and the ground truth to the total number of tokens in the prediction:

$$\text{Precision} = \frac{\text{Number of shared tokens}}{\text{Total number of tokens in prediction}}$$

Recall is the ratio of the number of shared tokens between the prediction and the ground truth to the total number of tokens in the ground truth:

$$\text{Recall} = \frac{\text{Number of shared tokens}}{\text{Total number of tokens in ground truth}}$$

The **F1 score** is the harmonic mean of precision and recall:

$$\text{F1} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

If Precision + Recall $= 0$, the F1 score is defined as 0.0.

Both EM and F1 score are crucial for a comprehensive evaluation, as EM provides a strict measure of correctness, while F1 score accounts for partial correctness and is often more indicative of human like understanding in cases of slight variations.

B Appendix B: Previous Assignment Context

This project leverages concepts and techniques developed in previous assignments of the course, specifically from Assignment 1 Named Entity Recognition (NER) and Assignment 2 Machine Learning with Word Embeddings.

B.1 Assignment 1: Named Entity Recognition (NER)

Assignment 1 focused on implementing and comparing various NER tools. While a dedicated NER tool was not directly integrated as a primary answer extraction component in this projects factoid QA, the understanding gained from working with entity types and their classification was foundational. The experience with identifying and categorizing entities like PERSON, DATE, ORG, LOC directly informed the design of the `QuestionClassifier` in section 3.3. The objective in Assignment 1 was to predict and evaluate these entity types, a skill directly applied here for guiding answer extraction.

B.2 Assignment 2: Machine Learning with Word Embeddings

Assignment 2 involved creating and evaluating various word embedding models from TF-IDF, Word2Vec, Doc2Vec, BERT embeddings and then using them as features for machine learning classification tasks for predicting movie popularity. This assignment provided crucial experience in, understanding how models like `SentenceTransformer`, a BERT-based embedding model, create rich semantic representations of text. This knowledge was directly applied in the `QuestionClassifier` and the `Semantic Re-ranking Ensemble` in section 3.7 and 3.3 to convert questions and answers into comparable vector spaces. Familiarity with `scikit-learn` classifiers like `RandomForestClassifier` and the principles of training/testing splits to prevent data leakage, and also the choice of random forest which showed the most promise in predictive accuracy. These practices were directly transferred to the `QuestionClassifier` module. The evaluation of classification models using accuracy, precision, and recall, as practiced in Assignment 2, formed the basis for evaluating the question and entity type prediction performance in this project. Thus, regarding the effectiveness of different embedding strategies and the robustness of various ML classifiers directly informed the design and implementation choices made for this comprehensive QA system.

C Appendix C: Core Code Snippets

This appendix provides selected code snippets from the jupyter notebook, highlighting the core functionalities and components of the Q&A system.

C.1 Data Loading: `load_collection` Function

The `load_collection` function is responsible for parsing the input JSON file and organizing the data into structured `Topic` and `QA` objects.

```
def load_collection(path: str|Path) -> List[Topic]:
    with open(path, 'r', encoding='utf-8') as f:
        raw_data = json.load(f)
    topics = []
    for entry in raw_data:
        content = entry.get('text') or entry.get('context')
        qa_items = []
        for qa_dict in entry.get('qa', []):
            question_text = qa_dict.get('question', "").strip()
            answer_text = qa_dict.get('answer',  "").strip()
            q_type = qa_dict.get('type')
            q_entity = qa_dict.get('entity')
            qa_items.append(QA(
                question=question_text,
                answer=answer_text,
                type=q_type,
                entity=q_entity
            ))
        topic_id = str(entry.get('id', ""))
        topic_title = entry.get('title', "").strip()
        topics.append(Topic(
            id=topic_id,
            title=topic_title,
            text=content,
            qa_list=qa_items
        ))
    return topics
```

Listing 1: `load_collection` function

C.2 Baseline: `random_word` Function

This function implements the simple random word selection baseline.

```
def random_word(text: str) -> str:
    words = re.compile(r"\\b\\w+\\b").findall(text)
    return random.choice(words)
```

Listing 2: `random_word` function

C.3 Question Classifier: `QuestionClassifier` Class

The `QuestionClassifier` class trains and predicts question types and factoid entity types using SentenceTransformers and RandomForests.

```
class QuestionClassifier:
    def __init__(self, sentence_embedder: SentenceTransformer):
        self.sentence_embedder = sentence_embedder
        self.type_classifier =
                        RandomForestClassifier(random_state=SEED)
        self.entity_classifier =
                            RandomForestClassifier(random_state=SEED)
```

```

    def train(self, qa_items: List[QA]) -> None:
        questions = [item.question for item in qa_items]
        types = [item.type for item in qa_items]
        feature_matrix = self.sentence_embedder.encode(questions)
        self.type_classifier.fit(feature_matrix, types)

        factoid_idxs = [i for i, t in enumerate(types)
                                    if t == "factoid"]

        if factoid_idxs:
            factoid_features = feature_matrix[factoid_idxs]

            factoid_entities = [qa_items[i].entity
                                        for i in factoid_idxs]

            self.entity_classifier.fit(factoid_features,
                                               factoid_entities)

    def predict(self, question: str) -> Tuple[str, Optional[str]]:
        features = self.sentence_embedder.encode([question])
        q_type = self.type_classifier.predict(features)[0]
        entity = (
            self.entity_classifier.predict(features)[0]
            if q_type == "factoid" else
            None
        )
        return q_type, entity
```

Listing 3: `QuestionClassifier` class

C.4 Passage Retriever: `PassageRetriever` Class

The `PassageRetriever` class identifies the most relevant topic text for a given question using a CrossEncoder.

```
class PassageRetriever:
    def __init__(self, device: int = DEVICE):
        self.model = CrossEncoder("cross-encoder/ms-marco-MiniLM-L6-v2",
                                               device=device)

    def best_topic(self, question: str, topics: List[Topic]) -> int:
        pairs = [(question, topic.text) for topic in topics]
        scores = self.model.predict(pairs)
        return int(np.argmax(scores))
```

Listing 4: `PassageRetriever` class

C.5 Factoid QA: `BERTpipeline` Class

The `BERTpipeline` class captures multiple BERT models for extractive question answering.

```
class BERTpipeline:
    def __init__(self, device: int):
        self.pipelines: Dict[str, "transformers.Pipeline"] = {
            name: pipeline(
                task="question-answering",
                model=path,
                tokenizer=path,
                device=device,
                token=HF_TOKEN2 # optional if needed could be left out
                # mainly to test models that require auth like qwen model
            )
```

```
            for name, path in BERT_MODELS.items()
        }

    def answer(self, question: str, context: str)
                    -> Dict[str, Tuple[str, float, float]]:
        res = {}
        for model_name, qa_pipe in self.pipelines.items():
            start_time = time.time()
            output = qa_pipe(question=question, context=context)
            latency_seconds = time.time() - start_time
            answer_text = output.get("answer", "").strip()
            confidence_score = output.get("score", 0.0)
            res[model_name] = (answer_text, confidence_score,
                latency_seconds)
        return res
```

Listing 5: `BERTpipeline` class

C.6 Factoid QA: `LlamaQA` Class

The `LlamaQA` class utilizes the Llama-2 LLM for generative question answering.

```
LLAMA_QA_PROMPT = (
"""<s>[INST] <<SYS>>
You are an AI that answers questions from texts.
<</SYS>>
Given the following report, answer the
question with the shortest possible phrase.
## REPORT
{context}
## QUESTION
{question} [/INST]
##ANSWER
"""
)

class LlamaModelBase:
    def __init__(self, model_path: str, hf_token: str
                                | None, device: int):
        self.enabled = torch.cuda.is_available() and bool(hf_token)
        self.model = None
        self.tokenizer = None
        quant_cfg = BitsAndBytesConfig(
            load_in_4bit=True,
            bnb_4bit_quant_type="nf4",
            bnb_4bit_use_double_quant=True,
            bnb_4bit_compute_dtype=torch.bfloat16
        )
        self.tokenizer = AutoTokenizer.from_pretrained(model_path,
                                                token=hf_token)

        self.model = AutoModelForCausalLM.from_pretrained(
            model_path,
            device_map="auto",
            quantization_config=quant_cfg,
            token=hf_token,
            trust_remote_code=True
        ).eval()

    def _generate_text(self, prompt: str, max_new_tokens: int) -> str:
        if not self.enabled or self.model is None: return ""
        inputs = self.tokenizer(prompt, return_tensors="pt")
        inputs = {k: v.to(self.model.device) for k, v in inputs.items()}
        generated = self.model.generate(
```

```
            **inputs, max_new_tokens=max_new_tokens, do_sample=False,
            eos_token_id=self.tokenizer.eos_token_id
        )
        output_text = self.tokenizer.decode(generated[0],
                                skip_special_tokens=True)
        return output_text

class LlamaQA(LlamaModelBase):
    def __init__(self, hf_token: str | None, device: int):
        super().__init__(LLAMA_MODEL, hf_token, device)

    def answer(self, question: str, context: str,
                        max_new_tokens: int = 128) -> str:

        prompt_text = LLAMA_QA_PROMPT.format(context=context,
                                             question=question)

        output_text = self._generate_text(prompt_text,
                                          max_new_tokens)
        if "##ANSWER:" in output_text:
            return output_text.
                        split("##ANSWER:")[-1].strip()
        else:
            lines = [line.strip() for line in output_text.splitlines()
                                          if line.strip()]
            return lines[-1]
                        if lines else ""
```

Listing 6: `LlamaQA` class with prompt

C.7 Confirmation QA: `ConfirmationAnswerer` Class

The `ConfirmationAnswerer` class uses a CrossEncoder and dynamic thresholding to answer "Yes/No" questions.

```
class ConfirmationAnswerer:
    def __init__(self, threshold: float = 0.5, device: int = DEVICE):
        self.threshold = threshold
        self.cross_encoder =
                    CrossEncoder("cross-encoder/stsb-distilroberta-base",
                                 device=device)
        self.sentence_embedder = None

    def set_sentence_embedder(self, embedder: SentenceTransformer):
        self.sentence_embedder = embedder

    def answer(self, question: str, full_topic_text: str) -> str:
        sentences = sent_tokenize(full_topic_text)
        sentences = [s.strip() for s in sentences if s.strip()]

        if not sentences: return "No"

        if self.sentence_embedder:
            question_embedding = self.sentence_embedder.
                                                encode([question])
            sentence_embeddings = self.sentence_embedder.
                                                encode(sentences)

            similarities = cosine_similarity(question_embedding,
                                             sentence_embeddings)[0]
            best_sentence_idx = np.argmax(similarities)
            context_for_cross_encoder = sentences[best_sentence_idx]
        else:
            context_for_cross_encoder = full_topic_text
```

```
        score = self.cross_encoder.predict([(question,
                                             context_for_cross_encoder)])[0]
        return "Yes" if score >= self.threshold else "No"
```

Listing 7: `ConfirmationAnswerer` class

C.8 Ensemble Strategies

The ensemble strategies are integrated into the main evaluation loop. The logic behind this is `majority_vote` and semantic re-ranking based on confidence thresholds.

```
# Majority Vote utility function
def majority_vote(predictions: List[str]) -> str:
    return Counter(predictions).most_common(1)[0][0]

# Snippet from evaluation loop for ensemble logic
best_bert_answer = ""
best_bert_confidence = 0.0
if bert_outputs:
    # Find BERT model with highest confidence
    best_bert_tag, (best_bert_answer, best_bert_confidence, _) = \
        max(bert_outputs.items(), key=lambda kv: kv[1][1])

# Initial ensemble (Majority Vote)
ensemble_answer_orig = majority_vote([best_bert_answer,
                                      llama_answer or best_bert_answer])

# Semantic Re-ranking
semantic_rerank_answer = ""
LOW_CONFIDENCE_THRESHOLD = 0.35 # Defined earlier in evaluate function

if best_bert_confidence < LOW_CONFIDENCE_THRESHOLD:
    candidate_answers = {best_bert_answer}
    if llama_answer:
        candidate_answers.add(llama_answer)
    candidate_answers = [ans for ans in candidate_answers if ans.strip()]

    if candidate_answers:
        question_embedding = sentence_embedder.encode([qa.question])
        candidate_embeddings = sentence_embedder.
                                                encode(candidate_answers)
        similarities = cosine_similarity(question_embedding,
                                         candidate_embeddings)[0]

        best_candidate_idx = np.argmax(similarities)
        semantic_rerank_answer = candidate_answers[best_candidate_idx]
    else:
        semantic_rerank_answer = best_bert_answer
                                    # Fallback if no valid candidates
else:
    semantic_rerank_answer = best_bert_answer
                                # Use best BERT directly if high
                                    confidence
```

Listing 8: Ensemble strategy snippets