# The Large Language Model aided expert problem

Anonymous ACL submission


## Abstract

Large Language Models (LLMs) often falter in providing accurate responses to queries that demand up-to-date or context-specific information. Retrieval-Augmented Generation (RAG) addresses this limitation by incorporating a retriever to fetch relevant documents from databases or the Internet. However, RAG falls short when relevant information is unavailable, necessitating expert intervention—a process that is both costly and inefficient. This work introduces and addresses the *LLM-aided expert problem*, aiming to develop systems that progressively enhance their competence in answering queries while minimizing the need for expert input. We propose two decision-making strategies: (1) a classifier-based approach that employs threshold-based filtering to evaluate retrieved answers, and (2) a contextual bandit approach that models the decision to rely on retrieved answers or escalate to an expert as a two-arm bandit problem. Both methods utilize Pretrained Language Models for answer validation and reward estimation. We evaluate these strategies using a benchmark derived from the Quora Question Pairs dataset, demonstrating their effectiveness in reducing expert interventions while maintaining high accuracy. Our results highlight the potential of adaptive decision-making frameworks to enhance LLM reliability in dynamic query-answering environments.


## 1 Introduction

The advent of Transformers (Vaswani et al., 2017) has spurred the development of numerous Large Language Models (LLMs), such as BERT (Devlin et al., 2019) and GPT (OpenAI et al., 2024), revolutionizing the processing and analysis of written language. However, LLMs face a significant limitation:They often struggle to address user queries that demand up-to-date or context-specific information, as they lack real-time awareness and may provide outdated or inaccurate responses.

To address this challenge, Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) was introduced. RAG enhances interaction by integrating a Retriever to fetch relevant documents from databases or the Internet. These retrieved documents, along with the user's query, are processed by a generative Large Language Model to produce an answer. RAG has been shown to outperform purely generative methods, significantly reducing hallucinations (Lewis et al., 2020; Ji et al., 2023; Sadat et al., 2023; Manakul et al., 2023).The term hallucination refers to the generation of false, misleading, or nonsensical information that appears plausible but is not based on real data or facts.

While RAG provides a substantial improvement over purely generative methods, it is still insufficient in scenarios where no relevant information exists in the database or online. In such cases, to address a query, the system requires an expert to intervene and provide the correct answer. This query, answer pair is then stored for future reference. However, expert interventions are costly, and querying the expert unnecessarily should be minimized. This raises a fundamental question:

*How can we design an LLM-based system that progressively builds competence in answering queries while minimizing reliance on expert interventions?*

We define this challenge as the LLM-aided expert problem and formulate it as an online optimization problem. Specifically, the system starts with an empty database and processes queries sequentially. For each new query, a retriever searches the database for similar past queries and their corresponding answers. The system then evaluates the retrieved answers to determine the most likely correct response. Based on this assessment, it either provides the retrieved answer to the user or assigns the query to an expert when necessary.

In this work, we explore strategies to optimize the trade-off between model autonomy and expert involvement, aiming to develop a system that maximizes accuracy while minimizing expert interventions. Our main contributions are as follows:

(a) We develop two sequential decision strategies for the LLM-aided expert problem.
1. The classifier-based approach. This strategy assesses whether the retrieved answers are accurate enough to be used. A classifier, combined with an optimized thresholding procedure, determines whether the system should return the retrieved answer or escalate the query to an expert.
2. The contextual bandit approach. We formulate the problem as a two-arm contextual bandit. Here, the context corresponds to the incoming query and the retrieved query-answer pair. Selecting the first arm means returning the retrieved answer, while selecting the second arm means consulting the expert. We propose a bandit algorithm that estimates the expected reward of the first arm and makes decisions accordingly.

In both strategies, we leverage a large variety of pretrained language models — for answer assessment in the classifier-based approach and reward estimation in the bandit algorithm.

(b) Empirical evaluation and benchmarking. We evaluate our methods using a question-answering benchmark based on the Quora Question Pairs dataset (Wang et al., 2017). To facilitate experimentation, we develop an environment that allows testing various methods and LLM architectures. Our results demonstrate the effectiveness of our strategies, showing significant improvements over naive baseline approaches.

## 2 Related Work

In contemporary applications of NLP to real-world question-answering environments, where knowledge may reside in documentation, Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) has emerged as a pivotal technique. A RAG system comprises two main components: a Retriever and a Large Language Model. Retrievers excel at representing similar words and sentences closely in the embedding space, thereby understanding language patterns effectively.

Notable retriever models, such as Dense Passage Retrieval (DPR) (Karpukhin et al., 2020), Embeddings from bidirectional Encoder Representations (E5) (Wang et al., 2022), and General Text Embedding (GTE) (Li et al., 2023), leverage pretrained architectures like BERT (Devlin et al., 2019) to initialize encoders. These encoders $E$ are fine-tuned to ensure that the cosine similarity $cosine\_similarity(E(x), E(y))$ accurately captures the true relationship between the query $x$ and the passage $y$.

Recently, the NLP community has increasingly favored decoder architectures for creating embeddings (Springer et al., 2024; BehnamGhader et al., 2024), as these approaches have demonstrated superior performance over traditional encoder-based methods. Among these, Mistral-E5 (Wang et al., 2024a) stands out as the state-of-the-art LLM for text retrieval. Retriever models are commonly evaluated using benchmarks such as BEIR (Thakur et al., 2021) and MTEB (Muennighoff et al., 2022).

Several approaches have been proposed to enhance the quality of RAG systems. Self-RAG (Asai et al., 2023) improves quality through retrieval and self-reflection, training a single language model to adaptively retrieve passages on demand and generate and reflect on both retrieved passages and its own outputs using reflection tokens. This makes the language model controllable during inference, allowing it to adapt to diverse task requirements.

R3 (Ma et al., 2023) introduces a Rewrite-Retrieve-Read scheme to effectively retrieve necessary knowledge. This approach employs a read-and-rewrite LLM, where a trainable small LLM generates queries and is trained via reinforcement learning based on feedback from a larger reader LLM.

M-RAG (Wang et al., 2024b) presents a dynamic system that achieved improvements of 11%, 8%, and 12% in text summarization, machine translation, and dialogue generation, respectively. This system utilizes two contextual bandit agents. During generation, Agent-S selects the database partition to address questions, and the retriever fetches the most relevant document. A generative LLM then produces multiple summaries, and Agent-R identifies the best possible summary. The final response is generated by an LLM based on the summary and retrieved document. Only the agents are trained, modeled as Markov decision processes, and optimized using Deep Q-Network (DQN) (Mnih, 2013) with replay memory.

## 3 Quora Question Groups

In this section, we introduce the dataset used in our experiments to illustrate and clarify the LLM-aided expert problem. The dataset is derived from the

Quora Question Groups, a clustered subset of the Quora Question Pairs dataset (Wang et al., 2017). The Quora Question Pairs dataset comprises over 400,000 question pairs, each annotated with a binary label indicating whether the questions are paraphrases of each other. Details of our dataset sampling process from the Quora Question Pairs dataset are provided in Appendix A.

A key concept in our experiments is the notion of a "question group." Two different questions belong to the same group if they can be addressed by the same answer. For example:

Q1: How do I erase my profile on Quora?
Q2: I can't get rid of this annoying account. Can someone help me?
A: To unsubscribe from Quora, you need to use a full-on browser and follow these steps: ...

From the Quora Question Pairs dataset, we extracted 7,365 questions along with their corresponding groups or answers. The dataset was split into training (66%), test (19%), and validation (15%) sets, ensuring no overlap of question groups across these subsets to assess the generalization capabilities of our models (see Table 1).

Figure 1 presents histograms of group sizes across the datasets. Group sizes range from 1 to 100, where a group size of 100 indicates a frequently asked question posed in various ways by at least 100 users. Groups with a size of one contain questions asked only once.

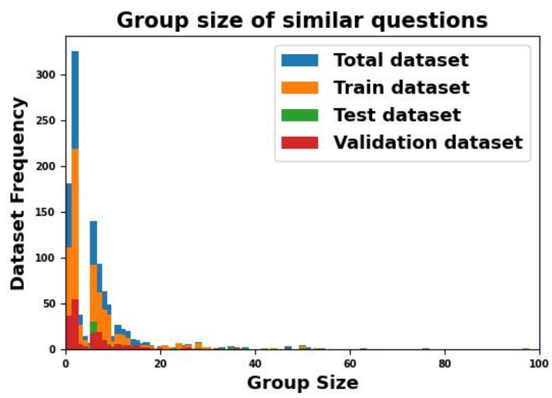


Figure 1: distribution of our groups in the final dataset.

## 4 LLM-Aided Expert System

In this section, we formally introduce the LLM-aided expert problem, outline various strategies for system design, and describe the training and testing procedures.

| | questions | groups | group size < 10 | group size = 1 |
|---|---|---|---|---|
| train | 4854 | 730 | 495 (68%) | 111 (15%) |
| test | 1424 | 181 | 120 (66%) | 33 (18%) |
| validation | 1087 | 189 | 115 (61%) | 37 (20%) |
| total | 7365 | 1100 | 730 (66%) | 181 (17%) |

Table 1: Dataset Statistics

### 4.1 The LLM-aided Expert Problem

The LLM-aided expert problem is an online decision-making problem where the system interacts sequentially with its environment. This interaction is illustrated in Figure 2.

**Environment.** In our setup, users sequentially submit queries to the system. The system can either respond directly or return an empty response ($\emptyset$), indicating the need for expert intervention. We assume the expert can always provide correct answers. Each expert response is stored in the system's database for future reference.

**System components.** The system comprises a database, a retriever, and an agent. The database stores queries and their corresponding expert-provided answers. The retriever identifies the $k$ most similar queries from the database to the incoming query. These $k$ queries and their answers are passed to the agent, which decides whether to use one of these answers or engage the expert. If the agent uses a previous answer, the new query-answer pair is not stored to avoid potential contamination of the database with incorrect information. However, every expert engagement results in storing the new query-answer pair in the database.

**Sequential interaction.** Initially, the database $\mathcal{B}_0$ is empty. In each round $t \geq 1$, a user generates a query $Q_t^R$ with an unknown correct answer $A_t^R$. The retriever identifies $k$ similar queries, and based on these, the agent either selects a corresponding answer or consults the expert. In practice, the agent will primarily engage the expert early on when the database is sparse. Over time, as the database grows, the system will handle most queries independently. However, novel queries may still necessitate expert intervention at any stage.

**Rewards and performance metrics.** The system receives a reward of +1 for correctly answering a query without expert help and incurs a penalty of -10 for incorrect answers. Engaging the expert results in a -1 penalty to discourage unnecessary consultations. Incorrect answers are heavily penal-

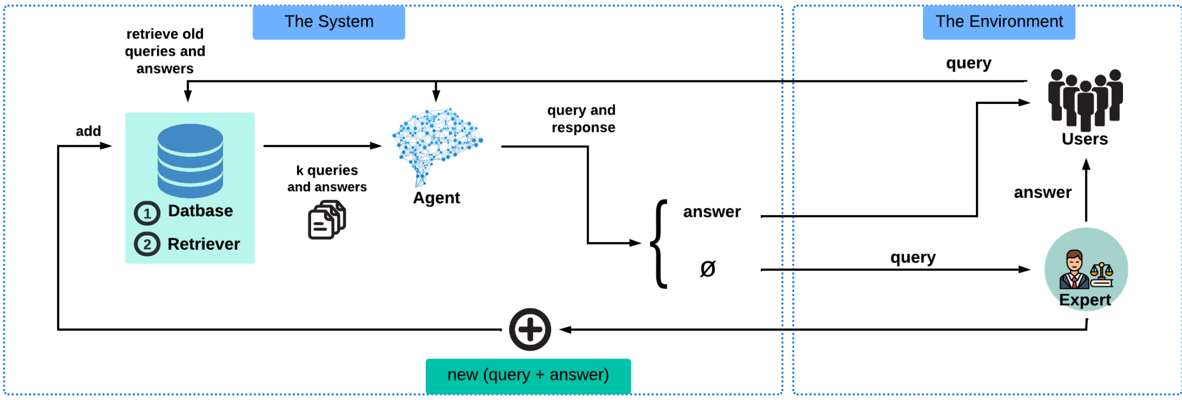


Figure 2: The LLM-aided expert system, and its interaction with the environment.

ized to prioritize accuracy over unnecessary expert engagement. The objective is to design an agent that maximizes cumulative rewards over a given number of rounds, starting with an empty database. Performance is also evaluated based on the number of incorrect answers and unnecessary expert engagements.

### 4.2 Agent design approaches

The agent processes incoming queries and the $k$ similar query-answer pairs provided by the retriever to either select an answer or request expert assistance. We present two approaches for designing the agent: the classifier and the contextual bandit approaches. These are compared against a baseline method where the retriever selects the answer.

**Baseline: the retriever.** The retriever is a static component of the system that remains untrained throughout the experiments. It identifies the $k$ most similar queries to the incoming query by leveraging query embeddings and calculating cosine similarity between the incoming query and database queries. The retriever can also function as an agent by selecting an answer based on these similarities: it returns the answer with the highest similarity to the incoming query if this similarity exceeds a predefined threshold. If no query in the database meets the similarity threshold, the retriever consults the expert. During the validation phase, the threshold is tuned to optimize performance.

**Classifier approach.** In the classification approach, the agent examines the incoming query and the $k$ similar query-answer pairs provided by the retriever. Each query-answer pair $(q_{\text{old}}, a_{\text{old}})$ is combined with the incoming query $q_{\text{new}}$ as follows:

$$[CLS]\ q_{\text{new}}\ [SEP]\ q_{\text{old}}\ [SEP]\ a_{\text{old}}\ [EOS] \quad (1)$$

where $[CLS]$ is the classification token, $[SEP]$ is the separator token, and $[EOS]$ is the end-of-sequence token. This sequence is fed into a language model to estimate the probability that $a_{\text{old}}$ is the correct answer to $q_{\text{new}}$. The agent returns the answer with the highest estimated likelihood if it 0.5. Otherwise, it consults the expert. The agent is initialized with a pretrained language model and fine-tuned using cross-entropy loss during training. The threshold is optimized during the validation phase.

**Contextual bandit approach.** In this approach, the agent's sequential decision problem is modeled as a two-arm contextual bandit. For each query-answer pair $(q_{\text{old}}, a_{\text{old}})$ returned by the retriever, the context is formed by combining the pair with the incoming query, as shown in equation (1). The first arm corresponds to returning $a_{\text{old}}$, while the second arm represents consulting the expert, with a known reward of -1. The reward of the first arm is estimated by passing the sequence through a pretrained language model, fine-tuned to minimize Mean Squared Error. This estimated reward is used to compute the index for the first arm. The final decision involves selecting the answer whose corresponding arm index is the highest, and exceeds a threshold, optimized during validation. If the threshold is not met by any query-answer pair, the second arm is played and the expert is called. The agent is initialized with a pretrained language model and fine-tuned during training.

**Algorithm 1** Training round

Pick $(Q_t^R,\ A_t^R)$ from $D_{train}$
$k\_pairs = Retriever(Q_t^R,\ database)$
$\overline{A} = agent.answer(Q_t^R,\ k\_pairs)$
**if** $\overline{A}\ =\ \emptyset$ **then**
  $R_t = -1$ (penalty for engaging the expert)
  $database.update(Q_t^R,\ A_t^R)$
**else if** $\overline{A}\ \neq\ A_t^R$ **then**
  $R_t = -10$ (incorrect answer penalty)
**else**
  $R_t = 1$ (correct answer reward)
**end if**
$agent.update(Q_t^R,\ A_t^R, R_t, k\_pairs)$

Figure 3: A round of our agent's training.

### 4.3 Training and testing phases

The agent is trained by simulating the operational environment, starting with an empty database. During each training round, as outlined in Figure 3, a query-answer pair $(Q_t^R,\ A_t^R)$ is randomly selected from the training dataset $D_{train}$. The retriever identifies the $k$ most similar query-answer pairs from the database. The agent then assesses whether any of the retrieved answers can address $Q_t^R$. If the agent determines that none of the retrieved answers are suitable, it returns $\emptyset$, prompting the expert to address the query. Each expert engagement incurs a penalty of -1, and the new query-answer pair is added to the database. If the agent provides an answer $\bar{A}$, it is compared with the original answer $A_t^R$. Our experiments utilize a cluster-based dataset where questions are grouped by their corresponding answers. This allows for comparisons by matching the retrieved query group ID with that of the incoming query $Q_t^R$. A correct match rewards the agent with +1, while an incorrect match results in a substantial penalty of -10. The agent's model is updated at the end of each training round using all collected information. To ensure the agent performs well even with a sparse database, the database is periodically emptied approximately every 1,000 rounds.

During the test phase, the environment is simulated using the test dataset $D_{test}$. The agent aims to answer as many questions correctly as possible, minimize expert engagement, and maximize knowledge acquisition. Unlike the training phase, the database is not periodically emptied. We consider two scenarios: one where the agent is not updated during testing, and another where it is updated at the end of each round, mirroring the training phase.

## 5 Experiments

Next, we present the agents we experimented with, detail the experimental setup, and discuss the results obtained.

### 5.1 Experimental Setup

Our experiments involve deploying pretrained language models as agents and fine-tuning them to optimize system performance.

**Retrievers.** We experimented with several retriever models, including: E5[1] (Wang et al., 2022), GTE[2] (Li et al., 2023), NOMIC[3] (Nussbaum et al., 2024), E5_Mistral[4] (Wang et al., 2024a).

**Agents.** We utilized a variety of language models as agents, including: BERT[5] (Reimers and Gurevych, 2019), MiniLM[6] (Wang et al., 2020), ALBERT[7] (Lan et al., 2019), DeBERTa_v3[8] (He et al., 2021), Mistral 7B[9] (Jiang et al., 2023), Llama 3 8B[10] (Dubey et al., 2024). A more detailed description of these language models is provided in Appendix C.

**Hyperparameter Tuning.** For each model, we conducted hyperparameter tuning to optimize performance. The hyperparameters experimented with are presented in Table 2. For the classifier approach, the 'class 0 weight' tunes the decision threshold used to decide whether the expert is consulted. Similarly the decision boundary in the bandit approach corresponds to the threshold used to assess the index of the first arm and to decide whether the second arm is played (the expert is consulted). The number of epochs corresponds to the number of times we go through the entire dataset. At the beginning of each epoch, the database is emptied, and the order in which queries are treated is randomized.

**Performance analysis.** We begin by evaluating our baseline, which is the retriever. We test various

[1] intfloat/e5-base
[2] thenlper/gte-base
[3] nomic-ai/nomic-embed-text-v1-unsupervised
[4] intfloat/e5-mistral-7b-instruct
[5] google-bert/bert-large-uncased
[6] microsoft/Multilingual-MiniLM-L12-H384
[7] albert/albert-large-v2
[8] microsoft/deberta-v3-large
[9] mistralai/Mistral-7B-Instruct-v0.3
[10] meta-llama/Llama-3.1-8B-Instruct

| Method | Hyperparameters |
|---|---|
| Retriever | decision threshold: from 0 to 1 with step size 0.05 |
| Classifier approach | learning rate: $\{1e-5, 2e-5, 3e-5\}$<br>$k$ during training: $\{1, 2, 5\}$<br>$k$ at test time: $\{1, 2\}$<br>training epochs: $\{5, 10, 15\}$<br>class 0 weights: $\{1, 5, 10\}$ |
| Bandit approach | learning rate: $\{1e-5, 2e-5, 3e-5\}$<br>$k$ during training: $\{1, 2, 5\}$<br>$k$ at test time: $\{1, 2\}$<br>training epochs: $\{5, 10, 15\}$<br>decision boundary: $\{-3, -2, -1, 0\}$ |

Table 2: Hyper parameters search space for all methods.

retriever models and select the best-performing one. This retriever is then combined with either our classifier or bandit agent. During testing, we consider two scenarios: one where the agent is not updated, and another where the agent is further fine-tuned.

Our experiments revealed that performance is significantly influenced by the sequence in which questions are presented. To ensure an unbiased estimation of validation and test performance, we conducted 10 validation iterations for each hyperparameter combination and 10 test iterations, shuffling the dataset in each iteration.

The results are presented in Table 3. Specifically, we report the mean and variance of the number of incorrect responses, the average number of unnecessary expert engagements, macro F1 score, weighted F1 score, and reward. Unnecessary expert engagement occurs when the answer is already in the database, but the expert is consulted nonetheless. In Appendix D, we provide results for scenarios where the agents process the sequences (1) without including the answer from the database.

### 5.2 Results

**Retriever performance.** In Table 3, the retriever performance is evaluated as zero-shot performance since no training is applied to the retriever. The only parameter optimized is the decision threshold during the validation phase. The E5_Mistral model (Wang et al., 2024a) achieves the highest reward while minimizing unnecessary expert engagement. However, the GTE model (Li et al., 2023) makes the fewest mistakes. Despite this, GTE's F1 score, accuracy, and reward are lower due to a higher frequency of unnecessary expert engagements. From these preliminary experiments, we observe that the highest reward is achieved when the agent effectively balances between incorrect responses and unnecessary expert engagements. Given that E5_Mistral (Wang et al., 2024a) was identified as the best retriever model for our problem, we utilize it to retrieve similar questions in subsequent experiments.

It is worth noting that GTE and Nomic used a portion of the Quora dataset to train the supervised versions of their models, which could potentially affect their performance in our context. Nomic provides both supervised and unsupervised versions of their model. Although the supervised version achieves slightly better accuracy and F1 scores, the unsupervised version performs slightly better due to fewer incorrect responses.

**Agents' Performance with training-only updates.** When the model is updated solely during training, the bandit agent based on Mistral (Jiang et al., 2023) outperforms all other models, including the retriever baseline. Notably, despite DeBERTa_v3 (He et al., 2021) having a relatively smaller size (330M parameters compared to Mistral's 7B and Llama's 8B), the DeBERTa_v3-based agent performs nearly as well as the Mistral-based agent. This finding is significant because it underscores the potential of DeBERTa_v3 in scenarios where powerful GPUs, necessary for running larger models like Llama, are unavailable.

**Agents' performance with updates at test time.** Table 3 demonstrates that updating the model at test time significantly enhances performance. Notably, even the MiniLM-based agent (Wang et al., 2020), with only 30M parameters, outperforms the E5_Mistral retriever (7B parameters) and the Mistral bandit agent (Jiang et al., 2023) when the latter is tuned only during training. This highlights that a less complex model, trained on relevant queries, can outperform more sophisticated models optimized for different types of queries. The highest reward is achieved by the Mistral classifier, which scores 107 reward units below the theoretically optimal reward of 1062. This model responded incorrectly an average of 4 times and unnecessarily engaged the expert 16 times, which is impressive given the 1424 questions received during testing.

It is important to note that the optimal reward can

| | Incorrect responses | Unnecessary engaged the expert | $F_1$ weighted | $F_1$ macro | Accuracy | Reward |
|---|---|---|---|---|---|---|
| *Zero-shot retriever performance* | | | | | | |
| E_5 | 22 (±8) | 104 (±6) | 91% (±7e-5) | 85% (±1e-4) | 91% (±7e-5) | 581 (±94) |
| GTE | **4 (±2)** | 164 (4) | 89% (±9e-6) | 81% (±2e-5) | 89% (±1e-5) | 658 (±29) |
| Nomic$_{unsupervised}$ | 10 (±4) | 96 (±6) | 92% (±2e-5) | 87% (±5e-5) | 92% (±2e-5) | 715 (±40) |
| Nomic$_{finetuned}$ | 16 (±6) | 72 (±4) | 93% (±1e-5) | 89% (±4e-5) | 94% (±2e-5) | 705 (±62) |
| E5_Mistral | 17 (±8) | **28 (±1)** | **97% (±3e-5)** | **94% (±9e-5)** | **97% (±3e-5)** | **791 (±83)** |
| *Performance of agents with training-only updates* | | | | | | |
| *Classifier approach* | | | | | | |
| MiniLM | 14 (±) | 89 (±6) | 93% (±5e-5) | 87% (±1e-4) | 93% (±5e-5) | 682 (±87) |
| BERT | 12 (±5) | 74 (±5) | 94% (±2e-5) | 89% (±4e-5) | 94% (±2e-5) | 740 (±55) |
| ALBERT | **7 (±3)** | 81 (±4) | 94% (±1e-5) | 89% (±4e-5) | 94% (±2e-5) | 774 (±34) |
| DeBERTa_v3 | **7 (±7)** | 54 (±4) | 96% (±3e-5) | 92% (±8e-5) | 96% (±3e-5) | **811 (±52)** |
| Mistral | 15 (±6) | **23 (±5)** | **97% (±3e-5)** | **94% (±9e-5)** | **97% (±3e-5)** | 806 (±73) |
| Llama | 20 (±7) | 26 (±3) | **97% (±3e-5)** | **94% (±1e-4)** | 95% (±3e-5) | 744 (±79) |
| *Bandit approach* | | | | | | |
| MiniLM | 15 (±11) | 80 (±7) | 93% (±2e-4) | 88% (±3e-4) | 94% (±2e-4) | 691 (±122) |
| BERT | 17 (±9) | 71 (±5) | 94% (±7e-5) | 89% (±2e-4) | 94% (±7e-5) | 687 (±107) |
| ALBERT | 20 (±9) | 63 (±8) | 94% (±6e-5) | 89% (±1e-4) | 94% (±6e-5) | 668 (±98) |
| DeBERTa_v3 | **12 (±6)** | 53 (±4) | 95% (±3e-5) | 91% (±9e-5) | 95% (±3e-5) | 772 (±70) |
| Mistral | **12 (±6)** | **19 (±2)** | **98% (±2e-5)** | **96% (±6e-5)** | **98% (±2e-5)** | **847 (±69)** |
| Llama | **12 (±5)** | 20 (±3) | **98% (±1e-5)** | **96% (±3e-5)** | **98% (±1e-5)** | 845 (±54) |
| *Performance of agents with updates at test time* | | | | | | |
| *Classifier approach* | | | | | | |
| MiniLM | 16 (±5) | 9 (±2) | 98% (±1e-5) | 96% (±5e-5) | 98% (±1e-5) | 859 (±46) |
| BERT | 17 (±5) | 10 (±2) | 95% (±9e-6) | 95% (±5e-5) | 98% (±1e-5) | 849 (±44) |
| ALBERT | 19 (±8) | 8 (±2) | 98% (±4e-5) | 96% (±2e-4) | 98% (±4e-5) | 827 (±73) |
| DeBERTa_v3 | 12 (±4) | **7 (±2)** | **99% (±5e-6)** | **97% (±3e-5)** | **99% (±6e-6)** | 901 (±30) |
| Mistral | **4 (±1)** | 16 (±5) | **99% (±1e-5)** | **97% (±5e-5)** | **99% (±1e-5)** | **955 (±20)** |
| Llama | **4 (±1)** | 19 (±16) | 98% (±1e-4) | **97% (±4e-4)** | 98% (±1e-4) | 947 (±37) |
| *Bandit approach* | | | | | | |
| MiniLM | 4 (±1) | 67 (±4) | 95% (±9e-6) | 90% (±3e-5) | 95% (±1e-5) | 853 (±19) |
| BERT | 5 (±2) | 52 (±4) | 96% (±9e-6) | 92% (±3e-5) | 96% (±1e-5) | 871 (±22) |
| ALBERT | 4 (±1) | 48 (±5) | 96% (±1e-5) | 93% (±3e-5) | 96% (±1e-5) | 887 (±17) |
| DeBERTa_v3 | **3 (±1)** | 32 (±4) | 98% (±9e-6) | 95% (±3e-5) | 98% (±1e-5) | 935 (±18) |
| Mistral | 6 (±1) | 16 (±5) | 98% (±5e-6) | **97% (±2e-5)** | 98% (±4e-6) | 931 (±21) |
| Llama | 5 (±1) | **14 (±2)** | **99% (±3e-6)** | **97% (±1e-5)** | **99% (±3e-6)** | **949 (±16)** |
| *Optimal performance* | | | | | | |
| | **0 (±0)** | **0 (±0)** | **100% (±0)** | **100% (±0)** | **100% (±0)** | **1062 (±0)** |

Table 3: Performance of the retrievers (top rows), agents with training-only updates (middle rows), and agents with updates at test time (bottom rows), when deploying both questions and answers for the evaluation.

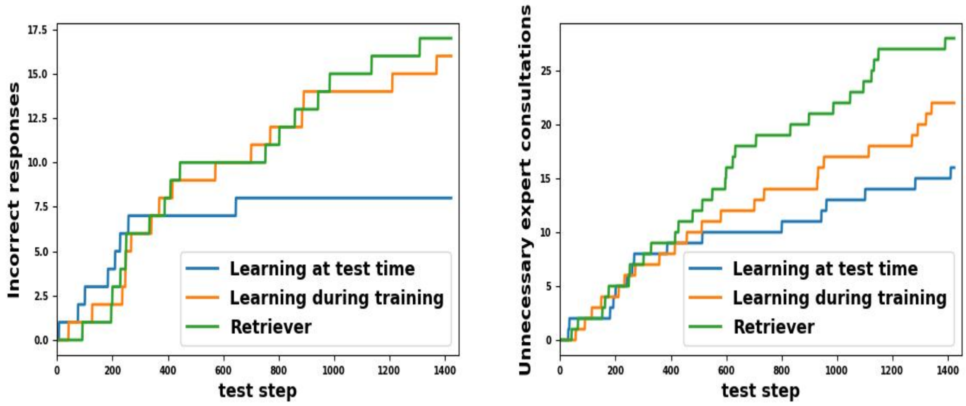


Figure 4: The number of incorrect answers (left) and of unnecessary expert consultations (right) vs the number of received queries at test time for the best models. The retriever is E5_Mistral; 'Learning at test time' corresponds to the Mistral classifier agent; 'Learning during training' to the Mistral bandit agent.

only be achieved if both the agent and the retriever are optimal. This requires all correct answers to be ranked within the top $k$ by the retriever and selected by the agent. In our experiments, we found that $k$ is typically 1 during testing, as retrieving more than one answer increases the likelihood of providing incorrect responses. Additionally, we observe that large language models generally outperform smaller ones across all experimental setups in this scenario.

**Dynamics under the best models.** Finally, we compare the dynamic behavior at test time of (i) the best retriever E5_Mistral model, (ii) the best agent with training-only updates, the Mistral bandit agent, (iii) the best agent with updates at test time, the Mistral classifier agent. Figure 4 presents one run of these agents at test time. The system starts with an empty database, and so naturally, the system returns incorrect answers and consults the expert often at the beginning. These events tend to discrease in rate as the system has seen more queries as the database grows large. The plots in Figure 4 really illustrates the advantage of updating the model at test time. Thanks to these updates, the model adapts to the new group of queries and in turn, makes almost no incorrect responses after receiving a few queries.

## 6 Conclusion and Future Work

In this paper, we introduced and addressed the LLM-aided expert problem, which arises in question-answering scenarios where initial information is lacking, necessitating expert intervention. We presented various approaches to develop agents that progressively enhance their competence in answering queries while minimizing the need for expert input. Our solutions demonstrated superiority over naive baseline methods.

Future work could focus on integrating generative large language models to dynamically generate responses based on prior interactions, marking the final step in deploying our system in real-world scenarios. Implementing this system will provide valuable insights into hallucination detection and enhance its practical applications. Additionally, we plan to explore the potential of applying our system to real-time dialogue scenarios, where responses will be dynamically generated based on previous questions and the evolving context of the conversation.

Additionally, deploying our system in real-world environments, such as customer support for new products or educational settings, would be valuable. In customer support, our system could handle a large volume of repetitive questions during product launches, enabling the testing of additional reinforcement learning methods. In education, our system could assist students by providing answers to frequently asked questions, ensuring accurate responses to maintain trust.

## 7 Limitations

Our approach relies on access to a substantial dataset of queries with corresponding correct answers during training. This dataset is essential for our agents to learn to evaluate the relevance of similar queries retrieved by the retriever and to back-propagate the loss from incorrect answers. Our experiments showed that to achieve high performance, especially with new types of queries in test data, the agent must be updated at test time. However, true answers are not available during testing.

In real-world scenarios, we expect users to provide feedback on the answers they receive, enabling the system to infer the validity of those answers. Real-world experiments are necessary to confirm this expectation. We have already tested our system at test time, assuming users complain when receiving a wrong answer. In this scenario, at test time, the agent was updated only when the user complained and when the expert was consulted. We observed no significant performance deterioration, compared to the scenario where true answers are known.

## References


Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi. 2023. Self-rag: Learning to retrieve, generate, and critique through self-reflection. *arXiv preprint arXiv:2310.11511*.

Payal Bajaj, Daniel Campos, Nick Craswell, Li Deng, Jianfeng Gao, Xiaodong Liu, Rangan Majumder, Andrew McNamara, Bhaskar Mitra, Tri Nguyen, et al. 2016. Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

Parishad BehnamGhader, Vaibhav Adlakha, Marius Mosbach, Dzmitry Bahdanau, Nicolas Chapados, and Siva Reddy. 2024. Llm2vec: Large language models are secretly powerful text encoders. *arXiv preprint arXiv:2404.05961*.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *ICLR*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *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)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Tianyu Gao, Xingcheng Yao, and Danqi Chen. 2021. SimCSE: Simple contrastive learning of sentence embeddings. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 6894–6910, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Pengcheng He, Jianfeng Gao, and Weizhu Chen. 2021. Debertav3: Improving deberta using electra-style pre-training with gradient-disentangled embedding sharing.

Pengcheng He, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. Deberta: Decoding-enhanced bert with disentangled attention. *arXiv preprint arXiv:2006.03654*.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *CoRR*, abs/2106.09685.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating hallucination in large language models via self-reflection. page 1827 – 1843.

Albert Q Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, et al. 2023. Mistral 7b. *arXiv preprint arXiv:2310.06825*.

Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite BERT for self-supervised learning of language representations. *CoRR*, abs/1909.11942.

Patrick S. H. Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*.

Zehan Li, Xin Zhang, Yanzhao Zhang, Dingkun Long, Pengjun Xie, and Meishan Zhang. 2023. Towards general text embeddings with multi-stage contrastive learning. *Preprint*, arXiv:2308.03281.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Xinbei Ma, Yeyun Gong, Pengcheng He, Hai Zhao, and Nan Duan. 2023. Query rewriting for retrieval-augmented large language models. *arXiv preprint arXiv:2305.14283*.

Potsawee Manakul, Adian Liusie, and Mark J.F. Gales. 2023. Selfcheckgpt: Zero-resource black-box hallucination detection for generative large language models. page 9004 – 9017.

Volodymyr Mnih. 2013. Playing atari with deep reinforcement learning. *arXiv preprint arXiv:1312.5602*.

Niklas Muennighoff, Nouamane Tazi, Loïc Magne, and Nils Reimers. 2022. Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar. 2024. Nomic embed: Training a reproducible long context text embedder. *Preprint*, arXiv:2402.01613.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, and et al. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Alec Radford and Karthik Narasimhan. 2018. Improving language understanding by generative pre-training.

Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence embeddings using Siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 3982–3992, Hong Kong, China. Association for Computational Linguistics.

Mobashir Sadat, Zhengyu Zhou, Lukas Lange, Jun Araki, Arsalan Gundroo, Bingqing Wang, Rakesh R. Menon, Md Rizwan Parvez, and Zhe Feng. 2023. Delucionqa: Detecting hallucinations in domain-specific question answering. page 822 – 835.

Jacob Mitchell Springer, Suhas Kotha, Daniel Fried, Graham Neubig, and Aditi Raghunathan. 2024. Repetition improves language model embeddings. *ArXiv*, abs/2402.15449.

Nandan Thakur, Nils Reimers, Andreas Rücklé, Abhishek Srivastava, and Iryna Gurevych. 2021. Beir: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *arXiv preprint arXiv:2104.08663*.

Aäron van den Oord, Yazhe Li, and Oriol Vinyals. 2018. Representation learning with contrastive predictive coding. *CoRR*, abs/1807.03748.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *CoRR*, abs/1706.03762.

Liang Wang, Nan Yang, Xiaolong Huang, Binxing Jiao, Linjun Yang, Daxin Jiang, Rangan Majumder, and Furu Wei. 2022. Text embeddings by weakly-supervised contrastive pre-training. *arXiv preprint arXiv:2212.03533*.

Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. 2024a. Improving text embeddings with large language models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 11897–11916, Bangkok, Thailand. Association for Computational Linguistics.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *CoRR*, abs/2002.10957.

Zheng Wang, Shu Teo, Jieer Ouyang, Yongjun Xu, and Wei Shi. 2024b. M-RAG: Reinforcing large language model performance through retrieval-augmented generation with multiple partitions. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1966–1978, Bangkok, Thailand. Association for Computational Linguistics.

Zhiguo Wang, Wael Hamza, and Radu Florian. 2017. Bilateral multi-perspective matching for natural language sentences. *CoRR*, abs/1702.03814.

## A Dataset Sampling

We outline the process of constructing our dataset using the Quora Question Pairs dataset. This dataset is represented as a graph where nodes correspond to questions, and an edge between two nodes signifies that the questions are equivalent. The graph is naturally divided into distinct groups, each representing a set of equivalent questions or connected components.

To create our dataset from the Quora Question Pairs dataset, we categorized the questions into five groups based on their frequency of occurrence:

1. Very Frequent: These questions belong to groups with more than 20 occurrences. They cover common and fundamental topics. The Quora dataset includes 107 groups of very frequent questions.

2. Frequent: Questions in this category belong to groups with 11 to 20 occurrences. The Quora dataset contains 324 frequent question groups.

3. Rare: These questions belong to groups with 6 to 10 occurrences. They focus on more specific areas. The Quora dataset includes 1,276 rare question groups.

4. Very Rare: Questions in this category belong to groups with 2 to 5 occurrences. They often pertain to specialized or niche topics. The Quora dataset contains 58,749 very rare question groups.

5. Unique: These are singular questions asked only once. They can be intriguing and unexpected. The Quora dataset contains 426,153 unique questions.

After categorizing the questions, we identified several groups containing paraphrased versions of the same question. To address this, we sampled 400 groups from each category, except for the "Frequent" and "Very Frequent" categories, which had fewer than 400 samples. We began with an empty dataset and, for each group, used our retriever to extract the 10 most common questions. We manually inspected these questions and checked if the new group matched any existing group in our database. If a match was found, we merged the new group with the existing one; otherwise, we added the new group to the database. This process was repeated twice.

Finally, we reviewed each group to ensure that all questions within a group could be answered with the same response. We then retrieved answers from Quora and revised those that did not adequately address the corresponding questions.

## B Implementation Details

Our experiments were conducted using the Transformers library. Our dataset consists of questions, group IDs, and the embeddings for each question. To avoid recalculating embeddings at every step, we precompute the embeddings at the beginning and utilize them throughout our experiments.

In each epoch of our training, we shuffle our dataset and traverse it sequentially. At each training step, we retrieve a question along with its embeddings and group IDs. Our memory consists of prior questions, organized in a matrix and indexed based on their respective positions. The matrix has a size of (number_of_previous_questions, embeddings_size).

By multiplying the embeddings of the new question with this matrix, we obtain a vector of size (number_of_previous_questions), which consists of the cosine similarity scores between the new question and the prior questions in our database. Subsequently, by performing a linear pass through this vector, we can retrieve the $k$ most similar questions to the new question based on our chosen metric.

For each previous question and its corresponding answer, we construct a triple sequence consisting of the new question, the old question, and the old answer. This sequence is then passed to our agent LLM, which can function either as a classifier or a contextual bandit. The agent's role is to determine whether the old answer is suitable for addressing the new question.

**The classifier approach.** The classifier takes the triple sequence as input and outputs the probability that the old answer can address the new question. If applicable, we return the most probable answers for the new question only if the probability exceeds 50%. We initialize our classifier using the standard *AutoModelForSequenceClassification* from the Transformers library, with *number_of_labels = 2*. The classifier is trained using cross-entropy loss.

**The Bandit approach.** The bandit takes the triple sequence as input and returns the expected reward. This reward can be $-10$ if the old answer does not address the new question, indicating that we should engage the expert, or 1 if it sufficiently addresses the new question, in which case we provide the previous answer as a response. Whether the old answer addresses the new question depends on the decision boundary, which is a hyperparameter we experiment with. This decision boundary is equivalent to the *no-response arm* and remains a constant value. We return the answer with the highest score if and only if the score exceeds the decision boundary. We initialize our bandit using the standard *AutoModelForSequenceClassification* from the Transformers library, with *number_of_labels* set to 1. The bandit is trained using mean squared error.

If the expert is engaged, we store the new question and its corresponding answer in the database. Otherwise, we do not store it to prevent contaminating the database with false information. At the end of each epoch, we reset the memory.

During the test and validation phases, we maintain the same experimental setup. The validation set is used to identify the best hyperparameters, while the test set is used to evaluate the performance of our model. If learning occurs during the test phase, we do not deploy the model trained on the validation set. instead, we use the checkpoint saved prior to the validation phase. For the learning rate, we apply a linear scheduler during training, while during testing, the learning rate is kept constant at $1e^{-5}$.

We conduct 10 independent evaluations and report the results as the average score along with the corresponding standard deviation or variance for each metric. For each evaluation, we reset the memory, shuffle the dataset, and reload the trained model from the specified checkpoint.

## C Pretrained Language models

In this section, we describe the Natural Language Processing models we deployed.

**E5 (Wang et al., 2022):** Embeddings from Bidirectional Encoder Representations (E5) is available in three versions. The first version is initialized with MiniLM (Wang et al., 2020), the second is initialized with BERT-base (Reimers and Gurevych, 2019), and the third is initialized with BERT-large. This model follows a bi-encoder architecture, where both the query and passage encoders are initialized with BERT. The training process consists of two stages. The first stage, called "unsupervised," uses a large number of unlabeled data, including title and passage pairs from Wikipedia, questions and answers from Reddit, and more. The InfoNCE contrastive loss (van den Oord et al., 2018) is employed to minimize the distance between related queries and passages, while maximizing the distance between unrelated queries and passages. The second stage involves training the model on high-quality human-annotated data, such as NLI (Gao et al., 2021), MS-MARCO (Bajaj et al., 2016), and Natural Questions (Karpukhin et al., 2020). During this stage, the model is trained with a loss that combines the KL divergence between the probability distribution of the label, as given by a cross-encoder teacher model, and the probability distribution generated by our E5 student model, along with the InfoNCE contrastive loss. This second stage further improves the model's performance on benchmarks such as BEIR (Thakur et al., 2021) and MTEB (Muennighoff et al., 2022).

**GTE (Li et al., 2023):** General Text Embedding (GTE) is available in three versions. The model's pretraining is divided into supervised and unsupervised phases. This paper introduces an improved contrastive loss, which is utilized in both phases of training. Additionally, they removed the KL objective, introduced a new sampling process, and expanded the dataset size during the supervised phase.

**NOMIC (Nussbaum et al., 2024):** Nomic is initialized from BERT and modified to address long-context retrieval. Nomic consists of 100 million parameters and supports a sequence length of up to 2048. Nomic's pretraining is divided into three stages. The first stage focuses on Masked Language Modeling to learn longer sequence representations. The subsequent stages are supervised and unsupervised, both employing the InfoNCE contrastive loss. This model was trained on a significantly larger dataset compared to the previous versions, encompassing both supervised and unsupervised phases. Nomic uses task-specific prefixes—such as search, search query document, classification, and clustering—to distinguish between the behaviors of each task. For the purpose of our work, we used the clustering prefix.

**E5_Mistral (Wang et al., 2024a):** This is the first unidirectional decoder architecture we use for our work. The model is initialized from Mistral 7B (Jiang et al., 2023) and consists of 7 billion parameters. The model takes as input the query $q^+$ and the task_definition and generates the instruction template:

$$q^+_{inst} = \text{'Instruct: \{task\_definition\} \textbackslash n Query: } \{q^+\}\text{'}$$

Where the task definition is:

'Given a web search query, retrieve relevant search queries that paraphrase the query.'

The query instruction template and the document instruction template are then passed to the LLM. We obtain the query and document embeddings, $h^+_q$ and $h^+_d$, from the last layer of the LLM at the [EOS] position. The model was trained on a large corpus of both original and synthetic data. The synthetic data was generated using advanced LLMs such as GPT-4.

**BERT (Reimers and Gurevych, 2019):** Bidirectional Encoder Representations from Transformers (BERT) has the same number of parameters as GPT (Radford and Narasimhan, 2018). However, unlike GPT, BERT uses bidirectional self-attention, whereas GPT uses constrained self-attention, where each token can only attend to the context to its left. The model was pretrained using Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). BERT's significant advancements over the state-of-the-art during this period sparked a major revolution in modern NLP, inspiring the development of numerous models. We use the large version of BERT, which consists of 304M parameters.

**MiniLM (Wang et al., 2020):** MiniLM is a simple and effective approach to compressing large Transformer-based pretrained models, referred to as deep self-attention distillation. The small model (student) is trained by closely mimicking the self-attention module, which plays a vital role in Transformer networks (Vaswani et al., 2017), of the large model (teacher). Specifically, they propose distilling the self-attention module from the last Transformer layer of the teacher, which is both effective and flexible for the student. MiniLM retains more than 99% accuracy on SQuAD 2.0 and several GLUE benchmark tasks using only 50% of the parameters and computational resources of the teacher model. The model consists of 66M parameters.

**ALBERT (Lan et al., 2019):** ALBERT is a more recent and efficient version of BERT that achieves state-of-the-art performance. In the original paper, two parameter-reduction techniques were proposed, resulting in significantly smaller architectures that enable much faster pretraining and fine-tuning. Additionally, the authors replaced the Next Sentence Prediction (NSP) objective with Sentence Order Prediction (SOP), as it was demonstrated to improve performance on GLUE, RACE, and SQuAD. We use the large version of this model, which consists of 18M parameters.

**DeBERTa_v3:** DeBERTa (He et al., 2020) is an improved version of BERT (Devlin et al., 2019) and RoBERTa (Liu et al., 2019) models, utilizing disentangled attention and an enhanced mask decoder. In the later version (He et al., 2021) of the model, it was discovered that using Replace Token Detection (RDT), originally deployed in ELECTRA (Clark et al., 2020), results in better performance than the Masked Language Modeling (MLM) objective. The paper also shows that vanilla embedding sharing in ELECTRA hurts training efficiency and model performance, as the training losses of the discriminator and generator pull token embeddings in different directions. We experiment with the large version of the model, which consists of 304M parameters.

**Mistral 7B (Jiang et al., 2023):** This is the first decoder model we utilize for our work. We experiment with both the vanilla version and the instruct version of the model. Instruct versions of LLMs are the vanilla models fine-tuned on instruction datasets. To fine-tune our model, we utilized LoRA (Hu et al., 2021). LoRA freezes the pre-trained model weights and injects trainable rank decomposition matrices into each layer of the Transformer architecture, significantly reducing the number of trainable parameters for downstream tasks. LoRA enables us to fine-tune LLMs consisting of billions of parameters on a single GPU. For all our models, we employed LoRA with a dimension of 16 and an alpha value of 8.

**Llama 3 8B (Dubey et al., 2024):** Llama 3 8B is the smaller model in the Llama 3 Herd of Models. This paper introduces a series of Llama models consisting of 8B, 70B, and 405B parameters. They also train two separate vision and speech encoders, as well as guard models for safe deployment of the LLMs. The model follows the standard Transformer architecture (Vaswani et al., 2017). The model was pretrained on 15T tokens, compared to the 1.8T tokens used for Llama 2. After pretraining, they utilized post-training methods like supervised fine-tuning (SFT), rejection sampling (RS), and direct preference optimization (DPO).

## D Additional Experiments

In this appendix, we provide results for scenarios where the agents process the sequences (1) without including the answer from the database. The results are presented in Table 4.

Our results demonstrate that the performance based only on questions is close to that achieved when considering both questions and answers. This can be attributed to the characteristics of our dataset. Specifically, according to the original DeBERTa_v3 paper (He et al., 2021), the model achieves an accuracy of 93% on the Quora Question Pairs dataset (Wang et al., 2017), leaving limited room for further improvement. This performance is slightly lower than that of the model trained solely during the training phase. However, the dynamic approach we implemented to create pairs using our retrievers clearly enhances the robustness of our fine-tuning process. By providing the most similar yet uncorrelated pairs, this method allows our agents to learn effectively and perform well in the most challenging scenarios.

It is important to note that Mistral (Jiang et al., 2023), fine-tuned on our relatively small dataset both as a classifier and as a bandit, outperforms the zero-shot performance of E5_Mistral (Wang et al., 2024a). This is particularly significant, as E5_Mistral (Wang et al., 2024a) has undergone considerably more training than our fine-tuned model. Similar to the performance based on both questions and answers, which we report in Table 3, we do not observe significant differences between the bandit and classification approaches. Moreover, it is clear that models with billions of parameters outperform smaller models. Finally, it is evident that, in this task, the DeBERTa_v3 model (He et al., 2021) demonstrates impressive performance, comparable to that of larger models.

| | Incorrect responses | Unnecessary engaged the expert | $F_1$ weighted | $F_1$ macro | Accuracy | Reward |
|---|---|---|---|---|---|---|
| | | | *Zero-shot performance* | | | |
| E_5 | 22 (±8) | 104 (±6) | 91% (±7e-5) | 85% (±1e-4) | 91% (±7e-5) | 581 (±94) |
| GTE | **4 (±2)** | 164 (4) | 89% (±9e-6) | 81% (±2e-5) | 89% (±1e-5) | 658 (±29) |
| Nomic$_{unsupervised}$ | 10 (±4) | 96 (±6) | 92% (±2e-5) | 87% (±5e-5) | 92% (±2e-5) | 715 (±40) |
| Nomic$_{finetuned}$ | 16 (±6) | 72 (±4) | 93% (±1e-5) | 89% (±4e-5) | 94% (±2e-5) | 705 (±62) |
| E5_Mistral | 17 (±8) | **28 (±1)** | **97% (±3e-5)** | **94% (±9e-5)** | **97% (±3e-5)** | **791 (±83)** |
| | | | *Performance of agents with training-only updates* | | | |
| | | | *Classifier* | | | |
| MiniLM | 13 (±7) | 108 (±7) | 92% (±5e-5) | 86% (±5e-5) | 93% (±4e-5) | 668 (±67) |
| BERT | 15 (±6) | 85 (±6) | 93% (±4e-5) | 88% (±1e-4) | 93% (±4e-5) | 687 (±55) |
| ALBERT | 15 (±8) | 102 (±6) | 92% (±5e-5) | 86% (±1e-4) | 92% (±4e-5) | 654 (±86) |
| DeBERTa_v3 | **7 (±7)** | 55 (±5) | 95% (±5e-5) | 90% (±1e-4) | 95% (±5e-5) | 714 (±93) |
| Mistral | 9 (±5) | **50 (±4)** | **96% (±3e-5)** | **92% (±7e-5)** | **96% (±3e-5)** | **811 (±63)** |
| Llama | 9 (±2) | 51 (±6) | **96% (±2e-5)** | **92% (±7e-5)** | **96% (±2e-5)** | 744 (±79) |
| | | | *Bandit* | | | |
| MiniLM | 10 (±7) | 92 (±7) | 93% (±5e-5) | 87% (±1e-4) | 93% (±5e-5) | 727 (±80) |
| BERT | **7 (±4)** | 91 (±5) | 94% (±2e-5) | 88% (±4e-5) | 93% (±2e-5) | 762 (±46) |
| ALBERT | 15 (±9) | 102 (±6) | 94% (±1e-5) | 86% (±1e-5) | 94% (±1e-5) | 654 (±86) |
| DeBERTa_v3 | 11 (±5) | 82 (±9) | 94% (±4e-5) | 88% (±1e-4) | 93% (±5e-5) | 738 (±62) |
| Mistral | 15 (±5) | **25 (±4)** | **97% (±1e-5)** | 95% (±1e-5) | **97% (±1e-5)** | **800 (±60)** |
| Llama | 19 (±7) | 26 (±4) | **97% (±2e-5)** | **97% (±5e-5)** | **97% (±1e-5)** | 756 (±76) |
| | | | *Performance of agents with updates at test time* | | | |
| | | | *Classifier* | | | |
| MiniLM | 21 (±7) | 14 (±3) | **98% (±2e-5)** | 95% (±9e-5) | **98% (±2e-5)** | 808 (±59) |
| BERT | 19 (±5) | 10 (±2) | **98% (±1e-5)** | 96% (±5e-5) | **98% (±1e-5)** | 825 (±44) |
| ALBERT | 19 (±5) | 13 (±8) | **98% (±2e-5)** | 95% (±1e-5) | **98% (±2e-5)** | 825 (±40) |
| DeBERTa_v3 | 16 (±4) | **6 (±2)** | **98% (±6e-5)** | **97% (±3e-5)** | **98% (±6e-5)** | 862 (±36) |
| Mistral | 15 (±5) | 24 (±4) | 97% (±2e-5) | 95% (±6e-5) | 97% (±2e-4) | **917 (±20)** |
| Llama | **4 (±1)** | 52 (±58) | 96%(±1e-3) | 93% (±3e-3) | 96%(±1e-3) | 799(±60) |
| | | | *Bandit* | | | |
| MiniLM | 4 (±1) | 83 (±7) | 94% (±6e-5) | 89% (±6e-5) | 94% (±2e-5) | 827 (±20) |
| BERT | 5 (±2) | 80 (±4) | 95% (±9e-6) | 89% (±3e-5) | 95% (±1e-5) | 821 (±16) |
| ALBERT | 5 (±2) | 57 (±5) | 96% (±1e-5) | 92% (±4e-5) | 96% (±1e-5) | 860 (±20) |
| DeBERTa_v3 | 8 (±2) | 35 (±4) | 97% (±1e-5) | 94% (±4e-5) | 97%(±9e-6) | 883 (±18) |
| Mistral | **4 (±2)** | **19 (±3)** | **98%(±6e-6)** | **97% (±2e-5)** | **98%(±6e-6)** | **941 (±20)** |
| Llama | 5 (±2) | 20 (±3) | **98%(±5e-6)** | 96% (±2e-5) | **98%(±5e-6)** | 938 (±16) |
| | | | *Optimal performance* | | | |
| | **0** (±**0**) | **0** (±**0**) | **100%** (±**0**) | **100%** (±**0**) | **100%** (±**0**) | **1062** (±**0**) |

Table 4: Performance of the retrievers (top rows), agents with training-only updates (middle rows), and agents with updates at test time (bottom rows), for various language models when deploying only questions for the evaluation.