COMILLAS
UNIVERSIDAD PONTIFICIA

ICAI

BACHELOR'S DEGREE IN MATHEMATICAL ENGINEERING AND ARTIFICIAL INTELLIGENCE

FINAL DEGREE PROJECT

ADAPTIVE OPTIMIZATION OF RAG SYSTEMS WITH BANDIT METHODS

Author: Jorge Ibinarriaga Robles

Supervisor: Jacobo Chaquet Ulldemolins

Madrid, June 2026

I declare, under my responsibility, that the Project submitted under the title

ADAPTIVE OPTIMIZATION OF RAG SYSTEMS WITH BANDIT METHODS

at the ICAI School of Engineering of Comillas Pontifical University in the academic year 2025/2026 is my own work, original and unpublished, and has not previously been submitted for any other purpose.

The Project is not plagiarized, either wholly or partially, and the information taken from other documents is duly referenced.

Signed: Jorge Ibinarriaga Robles Date: June 12th 2026

Submission of the project is authorized

THE PROJECT DIRECTOR

CHAQUET ULLDEMOLINS JACOBO - 51098165P

Firmado digitalmente por CHAQUET ULLDEMOLINS JACOBO - 51098165P
Fecha: 2026.06.12 10:31:19 +02'00'

Signed: Jacobo Chaquet Ulldemolins Date: June 12th 2026

COMILLAS
UNIVERSIDAD PONTIFICIA
ICAI

BACHELOR'S DEGREE IN MATHEMATICAL ENGINEERING AND ARTIFICIAL INTELLIGENCE

FINAL DEGREE PROJECT

ADAPTIVE OPTIMIZATION OF RAG SYSTEMS WITH BANDIT METHODS

Author: Jorge Ibinarriaga Robles

Supervisor: Jacobo Chaquet Ulldemolins

Madrid, June 2026

Acknowledgments

I would like to express my sincere gratitude to my thesis supervisor, Jacobo Chaquet Ulldemolins, who granted me this opportunity and gave me the idea of working on this cutting-edge project alongside BBVA upon the completion of my internship. Furthermore, I would also like to thank Marcos Galletero Romero, with whom I had the privilege of working and to whom I owe several valuable ideas and help throughout the project.

This work is dedicated to my parents, who have always been there supporting me throughout my degree, and my friends and family, who stood by me all along.

PONTIFICAL COMILLAS UNIVERSITY
School of Engineering (ICAI)
Bachelor's Degree in Mathematical Engineering and Artificial Intelligence

OPTIMIZACIÓN DE SISTEMAS RAG CON MÉTODOS BANDIT

Autor: Jorge Ibinarriaga Robles
Director: Jacobo Chaquet Ulldemolins
Entidad Colaboradora: Banco Bilbao Vizcaya Argentaria (BBVA)

RESUMEN DEL PROYECTO

Este proyecto explora la optimización dinámica de sistemas *Retrieval Augmented Generation* (RAG) mediante métodos de *Reinforcement Learning* (RL). Se aborda la selección automática de hiperparámetros que afectan directamente a la calidad de las respuestas. El problema se modela como un *multi-armed bandit* (MAB), un algoritmo que aprende iterativamente qué configuraciones (brazos) producen mejores resultados; equilibrando exploración y explotación. La evaluación se basa en un dataset de preguntas y respuestas, *SQuAD v2*, utilizando la similitud coseno para medir el rendimiento. Los resultados sugieren que ciertas configuraciones pueden adaptarse mejor a determinados conjuntos de preguntas, destacando el potencial de enfoques adaptativos en sistemas RAG.

Palabras clave: RAG; MAB; RL; LLMs; Question Answering (QA); embeddings; clustering

1. Introducción

El auge de la inteligencia artificial ha impulsado el uso de modelos de lenguaje en la industria. Sin embargo, estos modelos presentan limitaciones como la generación de respuestas incorrectas o "alucinaciones" en contextos específicos. Para mitigar este problema han surgido los sistemas RAG (Lewis et al., 2020), que permiten integrar documentos externos y mejorar la precisión de las respuestas. No obstante, su rendimiento depende significativamente de la configuración de hiperparámetros: un proceso costoso y normalmente dependiente de intervención humana.

En la práctica, los sistemas RAG se aplican en escenarios como asistentes internos en bancos o sistemas de soporte técnico basados en documentación. En estos casos, pequeñas variaciones en parámetros como el tamaño de los fragmentos (*chunk size*) o la temperatura de generación (*temperature*) pueden afectar significativamente la calidad de las respuestas.

2. Objetivos

En este contexto, en este trabajo se plantea la siguiente pregunta: ¿es posible automatizar la configuración de los sistemas RAG para optimizar su rendimiento en tareas de respuesta a preguntas sobre un documento dado?

Con el fin de abordar esta hipótesis y desarrollar una solución eficiente, se establecen los siguientes objetivos:

- Conseguir automatizar el proceso de configuración de hiperparámetros para un documento con un conjunto de preguntas y respuestas sobre ese documento.
- Encontrar un método de optimización de hiperparámetros que sea capaz de encontrar una configuración óptima de manera más eficiente que fuerza bruta mediante RL.
- Condicionar la configuración de hiperparámetros del RAG al tipo de pregunta realizada en lugar de usar una configuración fija. De esta forma, se mejora la calidad de las respuestas.

3. Descripción del sistema

Para abordar el problema, se ha diseñado una arquitectura que combina sistemas RAG mediante un enfoque de *multi-armed bandit* (MAB) (Auer et al., 2002), en la que un agente trata de encontrar la configuración con la mayor *reward* esperada de forma más eficiente que mediante una búsqueda por fuerza bruta.

Como se muestra en la Figura 1, el sistema parte de una arquitectura RAG clásica: *Indexer*, *Retriever* y *Generator*, la cual, dada una pregunta, genera una respuesta que es evaluada por una función de recompensa, *Rewarder*, en base a una respuesta de referencia (*ground truth*). Esta señal de recompensa es la que usa el agente MAB para seleccionar configuraciones de hiperparámetros (como *chunk size* o temperatura), balanceando entre exploración y explotación con el objetivo de maximizar la calidad de las respuestas.

Por otro lado, como se ilustra en la Figura 2, se introduce la fase de entrenamiento *offline* basada en *SQuAD v2* (Rajpurkar et al., 2018), donde las preguntas se agrupan mediante técnicas de *clustering* y se optimizan, para cada grupo, las configuraciones específicas del RAG. Posteriormente, en fase *online*, se encuentra para una nueva pregunta el *cluster* más cercano para obtener su configuración óptima, permitiendo aplicar la configuración previamente aprendida y mejorar la eficiencia y el rendimiento del sistema.

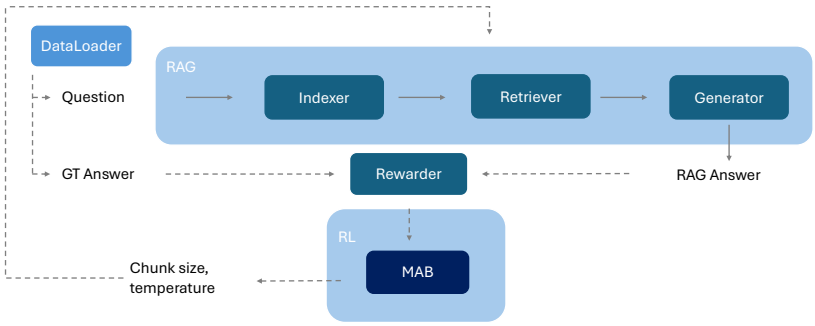


Figure 1: Arquitectura del sistema RAG con optimización mediante MAB.

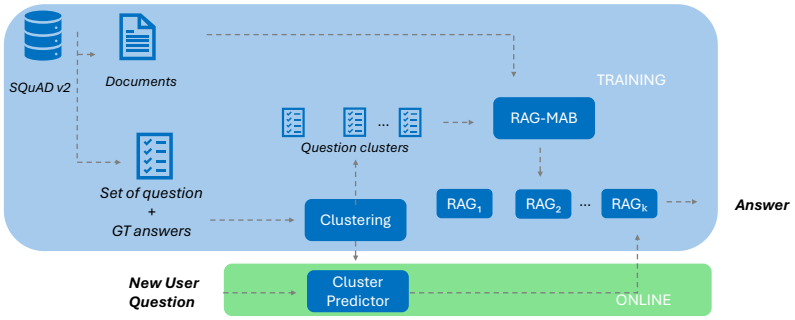
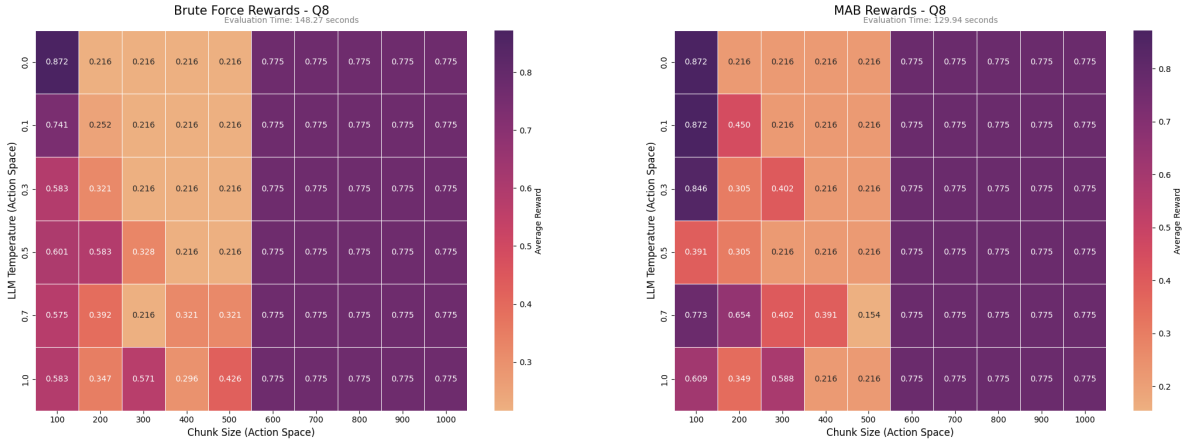


Figure 2: Pipeline de entrenamiento y despliegue del sistema.

4. Resultados

Los siguientes resultados sugieren que la optimización automática de hiperparámetros mediante MAB supera las configuraciones estáticas de RAG y permite una forma de optimización más eficiente que la búsqueda por fuerza bruta.

En la Figura 3, se puede observar la distribución de rewards inferida por fuerza bruta (a) y por un MAB (b). Se puede observar que son prácticamente idénticos.



(a) `heatmap_bf`: Brute Force

(b) `heatmap_mab`: Multi-Armed Bandit

Figure 3: Comparativa de heatmaps entre Brute Force y Multi-Armed Bandit.

Por otro lado, en la Figura 4 se observa cómo hay una diferencia significativa en los tiempos entre fuerza bruta y MAB; recalcando la eficiencia de los MAB. Además, en la Figura 5, se puede observar como el algoritmo MAB supera a un modelo *baseline* sobre un conjunto de preguntas.

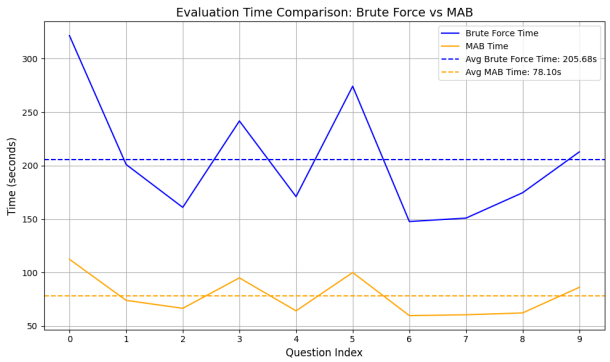


Figure 4: Comparativa del tiempo de evaluación entre Brute Force y MAB

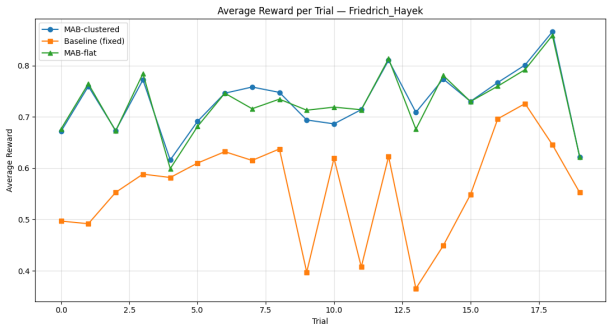


Figure 5: Comparativa de la reward media entre las variantes MAB y el baseline

5. Conclusión

A partir de estos resultados, podemos extraer dos conclusiones. Por un lado, el algoritmo MAB es capaz de replicar de manera eficiente la distribución de *Rewards* para un espacio de acciones definido por dos hiperparámetros (*chunk size* y *temperature*) en un tiempo significativamente menor que el tradicional por fuerza bruta. Por otro lado, el algoritmo MAB permite encontrar configuraciones óptimas en sistemas RAG que se adapten mejor a las características de las preguntas asociadas con un documento; mejorando el desempeño global de los sistemas RAG.

6. Referencias

[1] Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2–3), 235–256. https://doi.org/10.1023/A:1013689704352

[2] Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*.

[3] Rajpurkar, P., Jia, R., & Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

PONTIFICAL COMILLAS UNIVERSITY
School of Engineering (ICAI)
Bachelor's Degree in Mathematical Engineering and Artificial Intelligence

ADAPTIVE OPTIMIZATION OF RAG SYSTEMS WITH BANDIT METHODS

Author: Jorge Ibinarriaga Robles
Director: Jacobo Chaquet Ulldemolins
Collaborating Entity: Banco Bilbao Vizcaya Argentaria (BBVA)

ABSTRACT

This project explores the dynamic optimization of Retrieval Augmented Generation (RAG) systems through Reinforcement Learning (RL) methods. It addresses the automatic selection of hyperparameters that directly affect the quality of the answers. The problem is modeled as a multi-armed bandit (MAB), an algorithm that iteratively learns which configurations (arms) produce better results; balancing exploration and exploitation. The evaluation is based on a question-answering dataset, SQuAD v2, using cosine similarity to measure performance. The results suggest that certain configurations may adapt better to particular sets of questions, highlighting the potential of adaptive approaches in RAG systems.

Keywords: RAG; MAB; RL; LLMs; Question Answering (QA); embeddings; clustering

1. Introduction

The rise of artificial intelligence has driven the use of language models in industry. However, these models present limitations such as the generation of incorrect answers or "hallucinations" in specific contexts. To mitigate this problem, RAG systems (Lewis et al., 2020) have emerged, which allow integrating external documents and improving the accuracy of the answers. Nevertheless, their performance depends significantly on the hyperparameter configuration: a costly process that usually depends on human intervention.

In practice, RAG systems are applied in scenarios such as internal assistants in banks or technical support systems based on documentation. In these cases, small variations in parameters such as the size of the fragments (chunk size) or the generation temperature can significantly affect the quality of the answers.

2. Objectives

In this context, this work poses the following question: is it possible to automate the configuration of RAG systems in order to optimize their performance in question-answering tasks over a given document?

In order to address this hypothesis and develop an efficient solution, the following objectives are established:

- To automate the hyperparameter configuration process for a document with a set of questions and answers about that document.
- To find a hyperparameter optimization method capable of finding an optimal configuration more efficiently than brute force by means of RL.
- To condition the hyperparameter configuration of the RAG on the type of question asked instead of using a fixed configuration. In this way, the quality of the answers is improved.

3. Description of the system

To address the problem, an architecture has been designed that combines RAG systems through a multi-armed bandit (MAB) approach (Auer et al., 2002), in which an agent tries to find the configuration with the highest expected reward more efficiently than through a brute-force search.

As shown in Figure 6, the system starts from a classical RAG architecture: Indexer, Retriever and Generator, which, given a question, generates an answer that is evaluated by a reward function, Rewarder, based on a reference answer (ground truth). This reward signal is the one used by the MAB agent to select hyperparameter configurations (such as chunk size or temperature), balancing between exploration and exploitation with the goal of maximizing the quality of the answers.

On the other hand, as illustrated in Figure 7, the offline training phase based on SQuAD v2 (Rajpurkar et al., 2018) is introduced, where the questions are grouped through clustering techniques and the specific RAG configurations are optimized for each group. Subsequently, in the online phase, the closest cluster is found for a new question in order to obtain its optimal configuration, allowing the previously learned configuration to be applied and improving the efficiency and performance of the system.

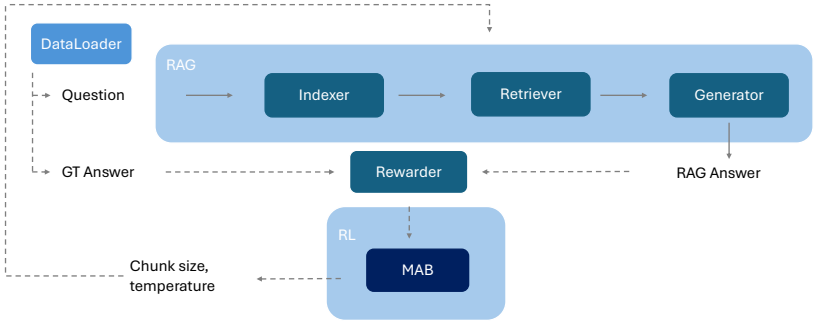


Figure 6: Architecture of the RAG system with MAB-based optimization.

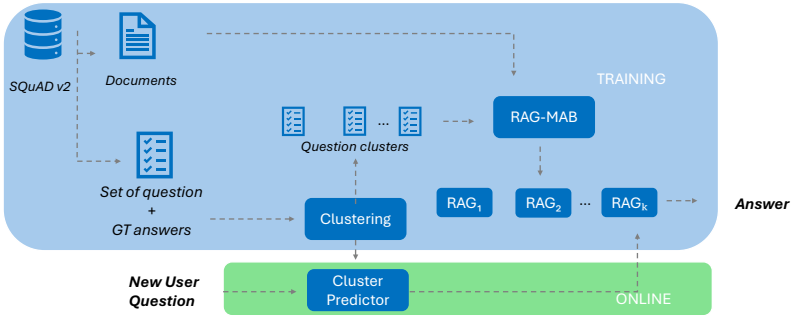
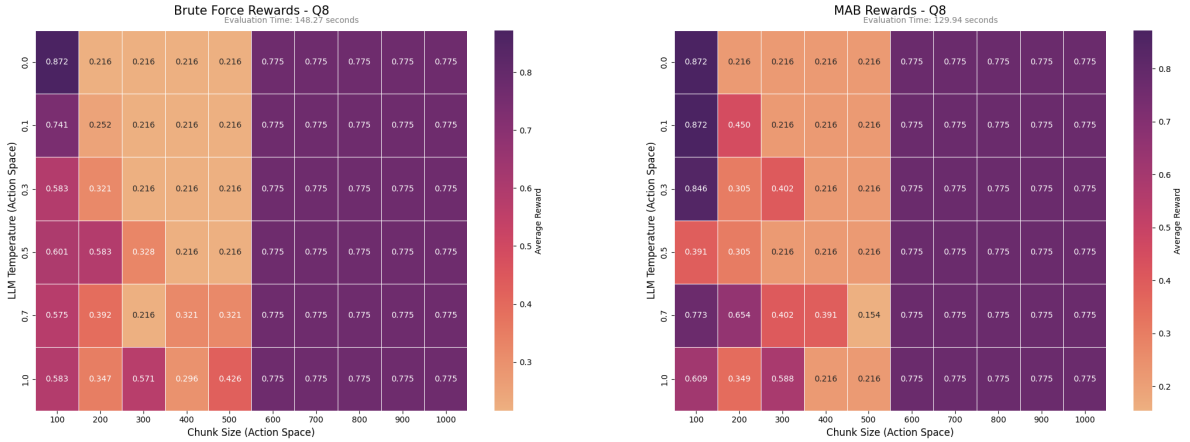


Figure 7: Training and deployment pipeline of the system.

4. Results

The following results suggest that automatic hyperparameter optimization through MAB outperforms static RAG configurations and enables a more efficient form of optimization than brute-force search.

In Figure 8, the distribution of rewards inferred by brute force (a) and by a MAB (b) can be observed. It can be observed that they are practically identical.



(a) `heatmap_bf`: Brute Force

(b) `heatmap_mab`: Multi-Armed Bandit

Figure 8: Heatmap comparison between Brute Force and Multi-Armed Bandit.

On the other hand, in Figure 9 it can be observed how there is a significant difference in the times between brute force and MAB; underscoring the efficiency of MABs. Moreover, in Figure 10, it can be observed how the MAB algorithm outperforms a baseline model over a set of questions.

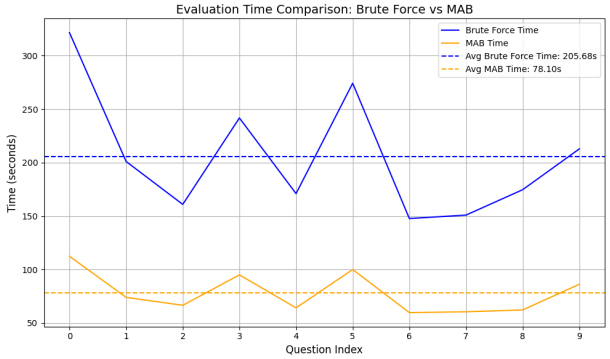


Figure 9: Comparison of evaluation time between Brute Force and MAB

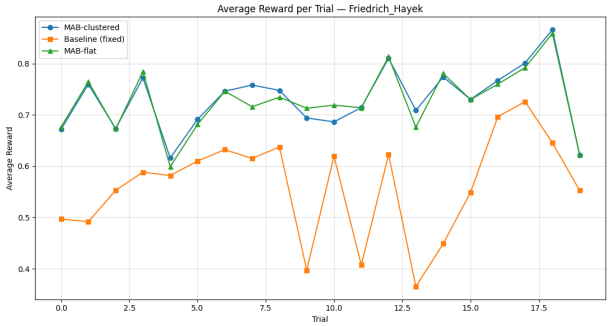


Figure 10: Average reward comparison between the MAB variants and the baseline

5. Conclusion

From these results, we can draw two conclusions. On the one hand, the MAB algorithm is able to efficiently replicate the distribution of Rewards for an action space defined by two hyperparameters (chunk size and temperature) in a significantly shorter time than the traditional brute-force search. On the other hand, the MAB algorithm makes it possible to find optimal configurations in RAG systems that better adapt to the characteristics of the questions associated with a document; improving the overall performance of RAG systems.

6. References

[1] Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2–3), 235–256. https://doi.org/10.1023/A:1013689704352

[2] Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*.

[3] Rajpurkar, P., Jia, R., & Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics*.

Contents

List of Figures

List of Tables

1 INTRODUCTION

Artificial Intelligence has undoubtedly become one of the main disruptive technologies in the 21st century. The emergence of generative AI through Large Language Models (LLMs) in recent years, such as OpenAI's ChatGPT or Google DeepMind's Gemini, has transformed the way people interact with information, and has significantly improved the efficiency of tasks such as software development, information retrieval, content generation and data processing. This has consequently had a major impact on several industries, affecting not only the technology sector, but also finance, healthcare, education, consulting and scientific research.

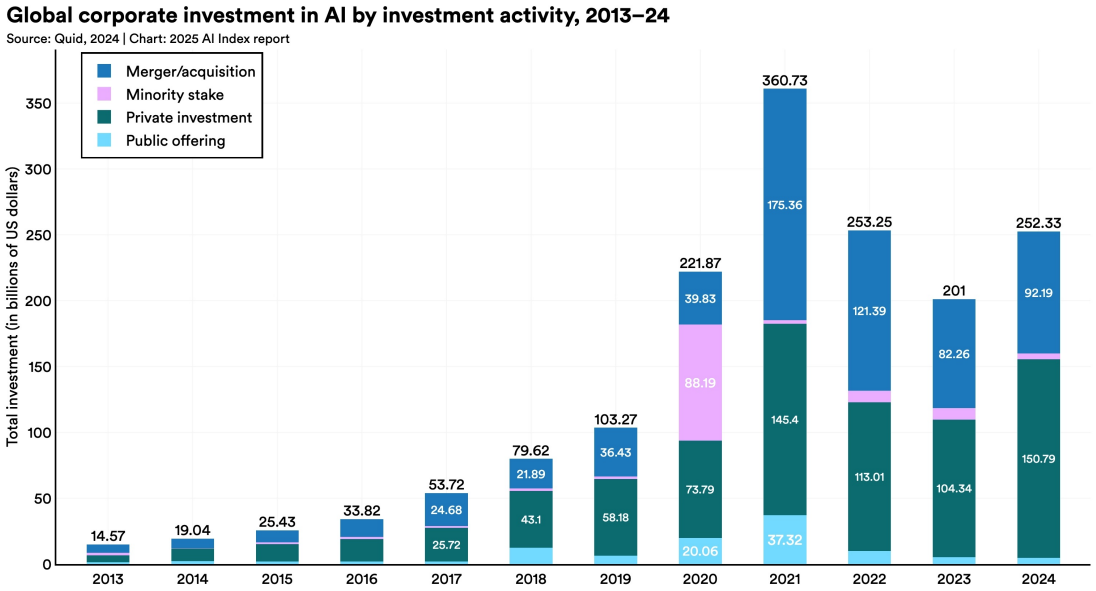


Figure 1: Global corporate investment in AI by investment activity (2013–2024). Source: Stanford AI Index Report 2025.

Thus, the economic and industrial impact of generative AI has been enormous. According to a report conducted by McKinsey & Company, generative AI could generate between $2.6 and $4.4 trillion annually across different industries, being one of the most significant technological revolutions since the internet. In the meantime, enterprise adoption and corporate investment in AI have accelerated rapidly. As illustrated in Figure 1, global corporate investment in AI reached more than $250 billion in 2024, particularly in private investment, reflecting the increasing importance of AI technologies across industries.

1.1 Context and Motivation

In spite of the impressive capabilities of LLMs when it comes to text generation, they still present major limitations. One of the most important challenges is hallucination, which occurs when the model produces responses that sound convincing but are factually incorrect or unsupported by evidence. This issue is becoming particularly problematic in those settings where inaccurate responses may yield severe consequences such as banking, medicine, technical support or the law. Recent studies have highlighted that hallucinations remain one of the key open problems in modern AI systems.

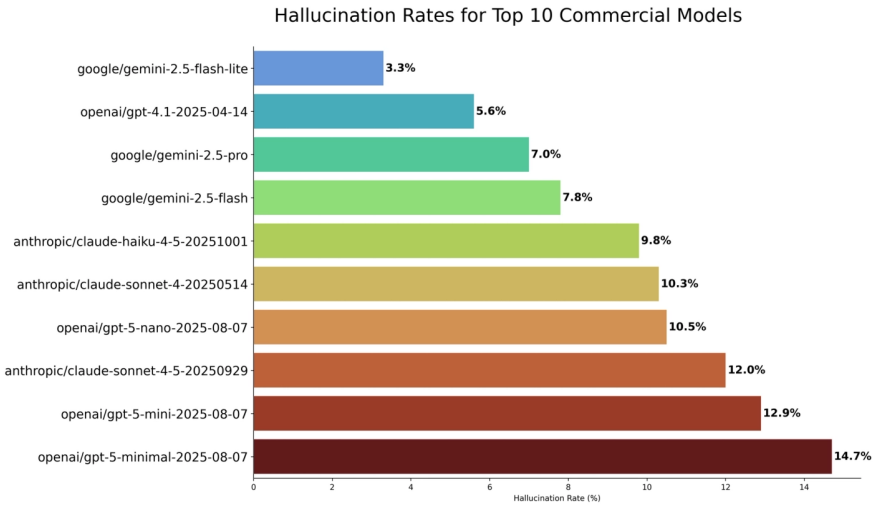


Figure 2: Hallucination rates across different Large Language Models. Source: Vectara Hallucination Leaderboard.

Figure 2 shows that even state-of-the-art commercial LLMs still face considerable hallucination rates, highlighting the difficulty of guaranteeing factual reliability in generation.

In order to mitigate this limitation, Retrieval Augmented Generation (RAG) systems have emerged as one of the main paradigms in the AI industry domain. RAG enables LLMs to access up-to-date and domain-specific information by combining document retrieval with text generation, instead of relying solely on pre-training and fine-tuning data. The work of Lewis et al., 2020 introduces the RAG architecture, demonstrating how document retrieval mechanisms can be integrated seamlessly with LLMs, improving factual accuracy and reducing hallucinations.

As a consequence, Retrieval-Augmented Generation (RAG) architectures are becoming increasingly common across several industries where access to factual and up-to-date information is critical. For instance, banks employ these systems to query internal regulations, analyze financial documentation, and support customer assistance services.

In particular, BBVA has developed a RAG-based information agent integrated into *Blue*, BBVA's generative AI assistant BBVA AI Factory, 2024. The system is designed to answer customer informational queries regarding banking products and general financial services. By

combining semantic retrieval mechanisms with Large Language Models (LLMs), the architecture enables more context-aware and reliable responses over internal banking documentation, reducing the risk of hallucinations and improving factual consistency.

Nevertheless, RAG systems introduce a new technical challenge: their performance depends significantly on the selection of hyperparameters such as chunk size, *top-k*, embedding strategy, or model temperature. Configuring these parameters remains one of the main bottlenecks when it comes to RAG system production. A recent Google Cloud engineering report on RAG optimization highlights how developers commonly optimize these systems through trial and error, repeatedly modifying retrieval and generation parameters while evaluating performance Google Cloud, 2024. Furthermore, the absence of standardized evaluation criteria makes it difficult to determine optimal configurations across different use cases, increasing the complexity of deploying RAG pipelines in real-world environments.

1.2 Objectives

In this context, this work studies the use of Reinforcement Learning techniques to dynamically optimize RAG systems. More specifically, the work studies whether Multi-Armed Bandit Methods, a class of Reinforcement Learning algorithms designed to balance exploration and exploitation under uncertainty, can efficiently adapt RAG hyperparameters.

In order to address the challenges previously discussed, the following objectives are proposed:

- **Automating RAG hyperparameter configuration.** Automate the process of configuring RAG hyperparameters for a given document and its associated question-answering task, reducing the need for manual trial-and-error experimentation.
- **Improving optimization efficiency through Reinforcement Learning.** Develop a hyperparameter optimization strategy capable of finding high-quality RAG configurations more efficiently than exhaustive brute-force approaches by using Reinforcement Learning techniques.
- **Adapting RAG configurations to the type of query.** Condition the RAG hyperparameter configuration on the type of question being asked instead of relying on a single static configuration, with the goal of improving the quality and robustness of generated responses.
- **Defining a consistent evaluation framework for RAG systems.** Introduce a systematic evaluation methodology based on cosine similarity metrics computed over semantic embeddings, using datasets containing documents, questions, and corresponding ground-truth answers in order to systematically compare different RAG configurations.

1.3 SDG Alignment

This work contributes to the United Nations Sustainable Development Goals (SDGs) United Nations, 2024 by leveraging Artificial Intelligence to improve information efficiency and technological infrastructure.

Particularly, the project aligns with the following:

- **SDG 4: Quality Education**

  Since this project aims to optimize the accuracy and groundedness of open-domain question-answering systems, it also contributes to providing more reliable and inclusive access to knowledge. The proposed methodology could improve the effectiveness of educational AI assistants and automated learning tools by reducing hallucinations and improving retrieval quality, ensuring that retrieved information is more accurate and contextually relevant.

- **SDG 9: Industry, Innovation, and Infrastructure**

  This project contributes to the development of more efficient, innovative, and scalable digital infrastructure based on Artificial Intelligence. By reducing the need for manual hyperparameter tuning and enabling systems to adapt configurations automatically, the proposed approach promotes more sustainable and efficient AI solutions that can be deployed in real industrial environments. Furthermore, the system may help organizations improve the use of AI technologies, increasing productivity and fostering technological innovation. In this sense, the project contributes to sustainable industrial development and intelligent digital infrastructure.

- **SDG 13: Climate Action**

  Recent studies have highlighted the significant environmental impact associated with Large Language Models, including high energy consumption, water usage, and greenhouse gas emissions MIT News, 2025. In this context, improving the efficiency of RAG systems may contribute to reducing unnecessary computational costs and redundant LLM calls. Since many RAG pipelines currently rely on extensive trial-and-error experimentation for hyperparameter tuning, the optimization framework proposed in this project may reduce the number of inefficient executions and unnecessary evaluations, thus indirectly contributing to more sustainable AI practices.

Figure 3: Sustainable Development Goals addressed by the project.

2 STATE OF THE ART

2.1 Related Work

This section reviews the main research areas related to this project.

2.1.1 RAG-RL: Advancing Retrieval-Augmented Generation via Reinforcement Learning and Curriculum Learning

The work of Huang et al., 2025 directly addresses the limitations presented by the use of static RAG pipelines by proposing RAG-RL, a framework that uses reinforcement learning and curriculum learning to improve answer generation. The motivation of this work lies in the improvement of Retrieval-augmented generation beyond retrieval and re-ranking, focusing on retrieving relevant contexts from multiple documents that require reasoning.

Instead of optimizing retrieval and re-ranking modules, this work suggests an answer generation model that can assess relevance of contexts with Reinforcement Learning using the Group Relative Policy Optimization (GRPO) algorithm. RAG-RL not only achieves performance gains in question answering, but also demonstrates improvement in document retrieval with both distractor-rich and gold-only settings.

Furthermore, the study shows how curriculum learning can affect model performance in post-training. In this approach, the model is first trained on easier samples, including only relevant documents, enabling the model to acquire more efficient and greater skills when it comes to citation, generation and reasoning. Training on more and more difficult curricula performs worse than min-max curricula, where the model shifts directly from easy to very hard samples. Overall, it has been found that mixing samples of varying difficulty can help the model learn more efficiently, identifying relevant contexts and generating citations.

Indeed, this work demonstrates the potential of reinforcement learning for improving RAG pipelines. However, this work focuses on the answer generation stage while the present project focuses on the retrieval configuration.

2.1.2 Beyond Static Retrieval: A Reinforcement Learning Framework for Dynamic and Adaptive RAG

In addition, Agnihotram et al., 2025 propose another Reinforcement Learning based framework, shifting traditional document retrieval toward a Markov Decision Process (MDP), thus tackling limitations of traditional static pipelines. This work attempts to reduce hallucinations and increase factual accuracy beyond conventional RAGs, adapting to rapidly evolving domains. Rather than using static, fixed selection methods, this approach allows the retrieval to dynamically learn and adapt over time through RL techniques. This process is framed with an MDP, enabling the system to make context-aware selections in both short and long term. RL allows RAG systems to benefit from dynamic adaptivity, context awareness, and long-term reasoning.

The work defines the retrieval process as a sequential decision-making process using the MDP framework. In this formulation, states represent the context including the query, documents retrieved so far, and similarity scores. Actions represent the decisions the retriever agent can make, such as selecting a document or reformulating a query. Finally, the rewards are signals received from document relevance, response quality, and user satisfaction. Q-learning, which learns the value of taking a certain action at a given state, is used to estimate which document is likely to lead to better generation.

Overall, Agnihotram et al., 2025 represent an important step towards dynamic and adaptive RAG systems introducing the use of Reinforcement Learning and MDPs. This is significantly related to the project as both aim to optimize RAG beyond static configurations. However, this work focuses on sequential RL for query reformulation and context selection using MDPs, whereas my project uses MAB techniques to dynamically adapt RAG retrieval configurations.

2.1.3 AutoRAG-HP: Automatic Online Hyper-Parameter Tuning for Retrieval-Augmented Generation

Yet another approach for creating adaptive RAG systems is the work of Fu et al., 2024. This work proposes a framework which addresses hyperparameter optimization through multi-armed bandit methods (MAB) for efficient exploration of the action space. The motivation of this work lies in the fact that the growing complexity of RAG pipelines requires careful tuning of configurable modules like retrieval or embedding. Exhaustive search techniques are often used in this context and are considerably expensive due to the high cost of LLM API calls.

Rather than relying on offline optimization methodology, this work proposes an online learning approach where each hyperparameter combination is represented as a bandit arm, and its reward is formulated as a linear combination of response accuracy and input token length, which penalizes the cost of LLM API calls.

The optimization algorithm used for the MAB selection is the Upper Confidence Bound, effectively balancing between exploration and exploitation. Each arm's reward is updated through a UCB step and eventually, the arm with the highest cumulative reward becomes the desired RAG configuration. Also, to address the exponential growth of possible combinations from tuning multiple hyperparameter configurations in MAB, authors introduce a two-level hierarchical MAB where higher-level bandits select which hyperparameters are to be tuned whereas low-level bandits search the optimal value within each parameter's search action space.

Altogether, this work demonstrates that MAB enables significantly faster adaptation as compared to other exhaustive search algorithms. Nevertheless, although both this work and the present project focus on MAB optimization of RAG configuration parameters using UCB, Fu et al., 2024 optimizes a broader set of modules, including prompt compression and embeddings across general QA benchmarks. Also, Fu et al., 2024 apply uniform configurations while this project introduces a query-aware adaptation mechanism through clustering.

All in all, these prior works demonstrate a growing interest in adaptive RAG systems with RL and MAB methods. However, none of these specifically explores query-aware adaptation, motivating the development of the present project toward context-sensitive RAG adaptation.

2.2 Technical Background

Multi-armed Bandits is a simplified reinforcement learning setting that does not require learning over multiple situations.

2.2.1 Reinforcement Learning

Reinforcement Learning is a branch of machine learning in which an intelligent agent takes actions in a dynamic environment to maximize a reward signal. This paradigm involves learning how to map situations to actions to maximize a reward. The agent does not know first hand which actions to take and must discover which actions yield the most reward by exploring them. Thus, the objective of the agent is to maximize cumulative reward over time (Sutton & Barto, 2018).

Unlike supervised learning, where the goal is to learn a set of labels from a knowledgeable supervisor in order to generalize situations not present in the training set, reinforcement learning focuses on learning how to maximize rewards through an agent interacting with its environment. One of the main challenges in reinforcement learning is the trade-off between exploration and exploitation. An agent may prefer actions that have been proven to be effective in the past in producing reward. However, in order to discover these actions, one needs to explore unseen actions that may not yield a high reward. Exploitation consists of selecting actions currently known to provide a high reward, while exploration involves trying out actions so as to make better selections in the future (Sutton & Barto, 2018).

RL has been proven to be successful in a wide variety of domains such as robotics, recommendation systems, and game play. In particular, in recent years, it has been tested as a mechanism for optimizing AI systems, including LLMs and even RAG architectures as we have seen in related work.

2.2.2 Multi-armed Bandits

Multi-armed Bandits are a simplified setting of reinforcement learning which do not involve learning how to act in more than one situation. This formulation avoids much of the complexity present in the full reinforcement learning setting as there are no state transitions or long-term sequences of decisions.

2.2.3 n-Armed Bandit Problem

The n-Armed Bandit Problem, or multi-armed bandit problem, is named after an analogy to a player facing a row of slot machines (one-armed bandits) who must decide which machines to play, as illustrated in Figure 4. Each action selection can be interpreted as pulling one

of the slot machine levers, while the rewards represent the payoff obtained from hitting the jackpot.

Figure 4: Slot machine analogy for the multi-armed bandit problem.

Formally, this problem is defined as a decision maker iteratively selecting one of n fixed choices (arms) where the selection of an arm does not affect the properties of the arm itself or other arms.

In the n-Armed bandit problem, each action has an expected reward conditioned on what action is selected. Let *value* be the expected reward of each action. Were the action values known, the n-armed problem would be trivial, as an agent would always select the action with the highest associated *value*. However, in this problem, which is therefore a type of Reinforcement Learning problem, it is unknown and is to be estimated. Also, this problem is a good example of the exploration-exploitation tradeoff dilemma. If the agent keeps estimates of the action *values*, there is always an action with the highest estimated *value*. This action is called a *greedy* action. If the agent selects this action, it would be *exploiting* its current knowledge of the action *values*. If, rather, the agent selects a *non-greedy* action, it would be *exploring* its current knowledge. Exploitation is the right choice to maximize the reward in the short-term, although exploration may yield a greater total reward in the long-term.

Formally, a stochastic multi-armed bandit can be defined as a set of reward distributions $\nu = (P_a : a \in \mathcal{A})$, where $\mathcal{A}$ is the set of available actions or arms (Sutton & Barto, 2018).

At each round $t \in \{1, \dots, n\}$, the learner selects an action $A_t \in \mathcal{A}$ and receives a reward $R_t \in \mathbb{R}$, sampled from the distribution P_{A_t}. Therefore, each arm has an associated reward distribution, but this distribution is unknown to the learner.

The objective is to maximize the cumulative reward obtained over n rounds:

$$S_n = \sum_{t=1}^{n} R_t$$

Since the reward distributions $\nu = (P_a : a \in \mathcal{A})$ are unknown, the learner must estimate

which arms provide higher rewards while interacting with the environment. The learner often relies on prior knowledge of ν, captured by an environment class $\mathcal{E}$, such that $\nu \in \mathcal{E}$.

When the environment class has a product structure:

$$\mathcal{E} := \{\nu = (P_a : a \in \mathcal{A}) : P_a \in \mathcal{M}_a \ \forall a \in \mathcal{A}\}$$

it is called unstructured: meaning playing arm a does not reveal any information about other arms' distributions.

Some typical choices of unstructured bandits $\mathcal{E}_B^k$ are Bernoulli bandits and Gaussian bandits, and broader non-parametric bounded-support bandits.

On the other hand, when the environment class $\mathcal{E}$ does not factor out, the problem is called structured. In this case, playing just one arm can reveal information from other arms. For instance, let:

$$\mathcal{A} \subseteq \mathbb{R}^d, \qquad \theta \in \mathbb{R}^d$$

$$\nu_\theta = (\mathcal{N}(\langle a, \theta \rangle, 1) : a \in \mathcal{A})$$

$$\mathcal{E} = \left\{\nu_\theta : \theta \in \mathbb{R}^d\right\}.$$

In this environment, the mean reward of an action is Gaussian and is given by $\langle a, \theta \rangle$. Thus, any arm's mean can be estimated by playing just d linearly independent actions (Sutton & Barto, 2018).

Let $\nu = (P_a : a \in \mathcal{A})$ be a stochastic bandit.

The expected reward, or mean reward, of arm a is defined as

$$\mu_a(\nu) = \int_{-\infty}^{\infty} x \, dP_a(x).$$

The optimal expected reward is then given by:

$$\mu^*(\nu) = \max_{a \in \mathcal{A}} \mu_a(\nu).$$

Furthermore, the regret of a policy π is given by:

$$R_n(\pi, \nu) = n\mu^*(\nu) - \mathbb{E}\left[\sum_{t=1}^{n} R_t\right] \tag{2.1}$$

The term $n\mu^*(\nu)$ represents the cumulative reward that would be obtained by choosing the optimal arm n times. The second term represents the actual cumulative reward obtained by the agent. Thus, minimizing regret is equivalent to maximizing the expected cumulative reward. In practice, regret is useful because it expresses the performance loss of the algorithm relative to the optimal arm. Following Sutton and Barto, 2018, the regret satisfies the following property:

Lemma 1. *The regret is always non-negative, and for every bandit ν, there exists a policy π, for which the regret vanishes.*

According to Lemma 1, zero regret can only be achieved through the optimal arm, which in practice is unknown. Therefore, the objective is to minimize regret over time. A desirable policy would be one with sublinear regret for all $\nu \in \mathcal{E}$:

$$\lim_{n\to\infty} \frac{R_n}{n} = 0$$

This implies that the average regret eventually converges to zero as the number of iterations increases; fewer suboptimal arms are selected over time, progressively converging toward the optimal arm. Consequently, the goal of a bandit problem is to identify the arm with the highest expected reward while minimizing the cost associated with exploring suboptimal policies.

2.2.4 Upper Confidence Bound Algorithm

A common optimization algorithm in MAB for arm selection based on reward function is the Upper Confidence Bound algorithm (UCB). This algorithm was formally analyzed by Auer et al., 2002 in *Finite-time Analysis of the Multiarmed Bandit Problem*, effectively balancing exploration and exploitation through an arm selection according to their upper confidence bounds. The algorithm is used for an unstructured bandit setting, where no prior information about relationships between arms is assumed. In this context, UCB relies only on the rewards observed for each individual configuration.

The core idea behind UCB is to select actions according to an optimistic estimate of their value. Rather than selecting the arm with the highest empirical mean reward (greedy), UCB adds an exploration bonus to each arm. This bonus is larger for those arms selected fewer times and becomes smaller and smaller as more arms are explored. Hence, naturally balancing exploration and exploitation; arms with high uncertainty are likely to be explored while arms with consistently high rewards are exploited.

Let A be the set of available arms, with $|A| = K$, where each arm $a \in A$ represents one possible action. At each round $t \in \{1, \ldots, n\}$, the learner selects an arm $A_t \in A$ and observes a reward R_t sampled from the unknown reward distribution associated with that arm. Let $N_a(t)$ denote the number of times that arm a has been selected before round t, and let $\hat{\mu}_a(t)$ be the empirical mean reward observed for arm a up to that point. After each arm has been selected at least once, UCB selects the arm:

$$A_t = \arg\max_{a\in A} \left(\hat{\mu}_a(t) + \sqrt{\frac{2\log t}{N_a(t)}} \right) \tag{2.2}$$

The first term, $\hat{\mu}_a(t)$, encourages exploitation by favoring arms with higher empirical rewards, whereas the second term,

$$\sqrt{\frac{2\log t}{N_a(t)}},$$

is an exploration bonus which decreases as $N_a(t)$ increases.

Algorithm 1 UCB1 algorithm, adapted from Auer et al., 2002

Require: Set of arms $A = \{1, \ldots, K\}$

Play each arm $a \in A$ once

Observe the corresponding rewards

Initialize $N_a(t)$ and $\hat{\mu}_a(t)$ for each arm $a \in A$

for $t = K+1, K+2, \ldots, n$ **do**

Select the arm

$$A_t = \arg\max_{a \in A} \left(\hat{\mu}_a(t) + \sqrt{\frac{2 \log t}{N_a(t)}} \right)$$

Play arm A_t and observe reward R_t

Update the number of selections:

$$N_{A_t}(t+1) = N_{A_t}(t) + 1$$

Update the empirical mean reward:

$$\hat{\mu}_{A_t}(t+1) = \hat{\mu}_{A_t}(t) + \frac{R_t - \hat{\mu}_{A_t}(t)}{N_{A_t}(t+1)}$$

end for

The implementation used to illustrate the behavior of the UCB algorithm is provided in Appendix B.

2.2.5 Clustering

Clustering is an unsupervised learning technique aimed at partitioning a set of data points into groups according to their similarity James et al., 2021. It is used in a wide variety of fields, particularly in Natural Language Processing where plain text is transformed into dense semantic embeddings through sentence transformers, so that texts with similar meaning are close in the same vector space. In this project, clustering is used because different types of questions may benefit from different RAG configurations.

There are several different clustering techniques. The algorithm used for clustering in this project is hierarchical agglomerative clustering, where each question is treated as an independent cluster and progressively merges the most similar clusters until all questions form a single cluster. The optimal number of clusters is automatically determined, which led to the selection of this algorithm, by the Silhouette method, which measures the quality of clustering determining how well each point lies within its cluster, yielding a score between -1 and 1, where higher values indicate better separated clusters (James et al., 2021). Besides,

the clustering prediction used for assigning new questions to an existing cluster is a weighted k-nearest neighbors predictor. The predictor computes cosine similarity between the new question and all previously clustered questions. Then, it selects the k most similar questions and assigns the new question to the cluster with the highest similarity score.

2.2.6 Retrieval-Augmented Generation

The core architecture used in this project is the Retrieval-Augmented Generation proposed by Lewis et al., 2020 in *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks*. This framework combines a parametric memory from a pre-trained seq2seq model with a non-parametric memory from a dense vector index over a large external corpus.

Given an input query x, the retriever selects a set of relevant documents z, and the generator produces the answer y conditioned on the query and the retrieved evidence.

Lewis et al., 2020 proposes two RAG formulations: one using the same retrieved documents for generation and another one using different retrieved documents.

In the first formulation, *RAG-Sequence*, the probability of generating an answer $y = (y_1, \ldots, y_N)$ given an input x is obtained by marginalizing over the top-k retrieved documents:

$$p_{\text{RAG-Sequence}}(y \mid x) \approx \sum_{z \in \text{top-}k(p_\eta(\cdot|x))} p_\eta(z \mid x) \prod_{i=1}^{N} p_\theta(y_i \mid x, z, y_{1:i-1}) \tag{2.3}$$

The retriever assigns $p_\eta(z \mid x)$ to each retrieved document z, while the generator estimates the probability of each output token conditioned on the query, the retrieved document and the previously generated tokens.

On the other hand, the second formulation, *RAG-Token*, allows the model to consider different retrieved documents when generating each token for the answer. Therefore, the model can rely on different pieces of retrieved information throughout the generation process:

$$p_{\text{RAG-Token}}(y \mid x) \approx \prod_{i=1}^{N} \sum_{z \in \text{top-}k(p_\eta(\cdot|x))} p_\eta(z \mid x) p_\theta(y_i \mid x, z, y_{1:i-1}) \tag{2.4}$$

Overall, both these formulations highlight that the quality of a RAG system depends on how effectively external information is retrieved and incorporated into the generation process.

3 MATERIALS AND METHODS

The resources on which the system was built and evaluated will be presented in this section. This includes what resources were taken and the rationale behind their selection.

3.1 Materials

- **Language Model.** For the generation module, `Llama 3.1 8B` Llama Team, AI @ Meta, 2024 was used as the local LLM instance through Ollama. A locally hosted, open-weight large language model was preferred over a commercial API, such as OpenAI, for several reasons. First, using the same LLM with the same weights for each generation call ensured reproducibility. Second, a significant number of generation calls was required, which would have implied extensive API usage, often leading to `Too Many Requests` errors.

- **Embedding Model.** Semantic representations are computed with the `all-mpnet-base-v2` sentence-transformer Reimers and Gurevych, 2019; Song et al., 2020. A single embedding model is used across the whole pipeline that requires semantic similarity: the retriever and the indexer module in the RAG architecture, the reward function, the clustering stage and the cluster predictor. This ensured semantic consistency across the whole pipeline, which was critical to ensure proper functionality.

- **Dataset.** The dataset used for experiments is `SQuAD v2` Rajpurkar et al., 2018, a reading-comprehension dataset that contains 100,000 questions on a set of Wikipedia articles. The dataset is divided into 536 articles containing questions with an associated context paragraph, the ground-truth answer, and the title of the article. This structure is particularly suitable for this work as the questions are grouped by articles, allowing each article to act as the corpus indexed by the RAG system, while the associated ground-truth answers can be compared with the generated RAG responses to compute the reward signal.

- **Hardware.** The system is run on a local machine with an `NVIDIA GeForce RTX 4070 Laptop GPU` with 8 GB of dedicated GPU memory. The system automatically uses a CUDA-capable GPU when one is available, falling back to CPU otherwise. The most computationally demanding parts of the pipeline, particularly in terms of GPU usage, are the language-model generation and re-indexing stages in the RAG system.

- **Technologies and Tools.** The project is implemented in Python and packaged by Poetry through `pyproject.toml` to ensure a reproducible environment. The RAG pipeline and LLM management are orchestrated by LangChain, responsible for the document loaders, as well as the embedding wrapper and the connectors to the language model. In addition, semantic vectors are produced with the `sentence-transformers` library and stored persistently in Chroma, used as the persistent vector store, while

Ollama loads the language model locally. The clustering and similarity stages are built on `scikit-learn` Pedregosa et al., 2011, and numerical operations rely on `NumPy`. The visualization of results is managed by `matplotlib` and `seaborn`. Finally, code quality and consistency practices are enforced with the Ruff linter.

3.2 Problem Formulation

Formal definition. Consider a fixed document (corpus) D along with a set of question–answer pairs:

$$\mathcal{Q} = \{(q_i, g_i)\}_{i=1}^{m},$$

where each q_i is a question that can be answered from D and g_i is its associated ground-truth answer.

A RAG system, as described above, is determined by a configuration c that fixes its hyperparameters; given a question q, the configured RAG answers:

$$\hat{y} = \mathrm{RAG}_c(q).$$

The problem that is presented in this work is to find, for a given document D, the optimal configuration $\hat{c}$ that maximizes the reward of the generated answers, in a more efficient way than by an exhaustive search over all configurations.

Action space. Amongst all hyperparameters of a RAG pipeline, this work optimizes the chunk size s, used to split the documents into chunks during the indexing stage, as well as the temperature τ of the language model during generation, while the remaining hyperparameters are kept fixed. This is because, as can be seen later in the results, varying chunk size and temperature is much more sensitive to changes as compared to other hyperparameters, and more interesting to optimize.

A configuration is therefore a pair $a = (s, \tau)$, and the set of admissible configurations forms the action space:

$$\mathcal{A} = \mathcal{S} \times \mathcal{T}, \qquad \mathcal{S} = \{100, 200, \ldots, 1000\}, \qquad \mathcal{T} = \{0.0, 0.1, 0.2, 0.3, 0.5, 0.7, 1.0\},$$

so that each arm $a \in \mathcal{A}$ corresponds to one concrete RAG configuration and $|\mathcal{A}| = 10 \times 7 = 70$.

Even though temperature is theoretically a continuous variable and chunk size can take several possible values, both were simplified using a smaller set of discrete values for simplicity. Thus, the action space is discrete and finite, which makes it well suited to a multi-armed bandit treatment.

Reward. The reward signal given by a generated answer $\hat{y}$ to the question q is given by the cosine similarity between its semantic embedding and the one of the ground-truth answer g.

Let $\phi(\cdot)$ be an arbitrary embedding model. The reward obtained when a configuration a is applied to the pair (q, g) is:

$$r(a; q, g) = \cos\big(\phi(\mathrm{RAG}_a(q)),\, \phi(g)\big) = \frac{\langle \phi(\mathrm{RAG}_a(q)),\, \phi(g)\rangle}{\|\phi(\mathrm{RAG}_a(q))\|\, \|\phi(g)\|}. \tag{3.1}$$

When $\tau > 0$, the language model is stochastic, and therefore, repeated evaluations of the same arm a on the same question q yield different answers, and hence the reward in Equation (3.1) is a random variable.

Multi-armed bandit formulation. This setting naturally fits the stochastic multi-armed bandit introduced above. Each action a is associated with an arm with an unknown reward distribution P_a, whose mean is given by:

$$\mu_a = \mathbb{E}\big[r(a; q, g)\big] \tag{3.2}$$

which represents the expected value of the answer quality using configuration a. Pulling an arm therefore corresponds to selecting a configuration, generating an answer and observing the resulting reward. Moreover, evaluating one configuration does not reveal information about the reward distribution of other arms, since the setting is unstructured. The reward distributions are assumed to be stationary and do not change during optimization.

The optimal configuration is the arm with the highest expected reward:

$$a^* = \arg\max_{a \in \mathcal{A}} \mu_a \tag{3.3}$$

which is unknown and must be estimated from interaction.

Optimization objective. The goal is not only to identify the optimal configuration a^*, but also to do so with as few evaluations as possible. This will directly reduce the number of language model calls and, consequently, the overall computational cost.

Query-aware configuration. Finally, the formulation above only selects a single configuration for the whole document. However, questions on the same document may be of different nature and benefit from different configurations. In order to address this, the question set $\mathcal{Q}$ is partitioned into groups according to semantic similarity, and a separate bandit problem is used to solve each group, so that the selected configuration becomes conditional on the query, making this problem a conditional optimization problem, rather than a global optimization problem. Note, however, that this does not become a conditional bandit problem; it remains a collection of independent bandits per group.

3.3 Solution Design

The solution proposed in this work combines two parts: a modular Retrieval-Augmented Generation system, as well as a multi-armed bandit optimization module. This is structured in two phases: an offline phase in which questions are clustered and the system learns the configuration that maximizes answer quality for each cluster, and an online phase that applies the learned configuration that best matches the new unseen questions.

The main goal is to automatically select the RAG configuration that best answers the questions associated with a given document conditioned on the type of question, rather than on a global setting. Consequently, exhaustive brute force over the action space will be replaced with a multi-armed bandit that concentrates evaluations on the most promising actions.

Offline phase. During training, the system is fed by a document along with a set of questions with associated ground-truth answers from the dataset. First, the questions are grouped into clusters according to their semantic similarity, then for each cluster, bandit optimization is run, producing an optimal configuration that maximizes the expected reward of the cluster. Formally, the question set $\mathcal{Q}$ is partitioned into K disjoint clusters:

$$\mathcal{Q} = \bigcup_{k=1}^{K} \mathcal{Q}_k, \qquad \mathcal{Q}_i \cap \mathcal{Q}_j = \varnothing \;\; (i \neq j),$$

and, for each cluster, the optimal configuration is the arm that maximizes the expected reward in the questions of the cluster:

$$a_k^* = \arg\max_{a \in \mathcal{A}} \; \mathbb{E}\big[\, r(a; q, g) \mid (q, g) \in \mathcal{Q}_k \,\big]. \tag{3.4}$$

Once the optimal configuration of each cluster has been obtained, a RAG instance is instantiated using its corresponding configuration. Therefore, the outcome of this phase is a set of cluster-specific RAG systems prepared for later use during inference.

Online phase. During inference, a new question is assigned to a cluster learned during training, according to its semantic similarity, and the RAG associated with that cluster, with the configuration a_k^*, is used to produce the answer. This allows the RAG configuration learned for each cluster to be reused without further evaluation.

RAG Architecture.

The RAG system is implemented from separate and independent modules, rather than a single block. Each module has a clear responsibility and is defined as an abstract interface: `BaseLoader`, `BaseIndexer`, `BaseRetriever` and `BaseGenerator`, and a single `RAG` class, which acts as the composition root that connects each module to build the whole system. The resulting software architecture is illustrated in Figure 5:

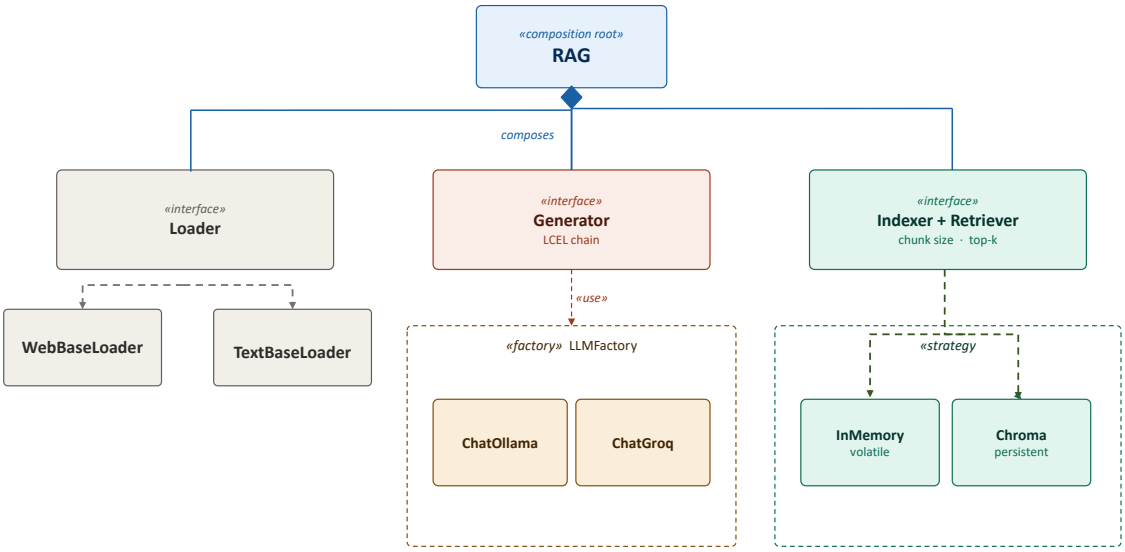


Figure 5: Design of the RAG system: a composition root that delegates to four abstract components, with the *Strategy* and *Factory* patterns.

In this system, the loader retrieves the raw document; the indexer splits it into chunks, whose size relies on the chunk size hyperparameter, embeds them and stores them in a vector store; the retriever then selects the top-k most relevant chunks for a given question; and finally the generator produces the final answer according to the other hyperparameter, temperature. The generation process is implemented using a LangChain Expression Language (LCEL) chain that takes the retrieved context together with the question into a prompt template, queries the language model, and parses its output. The prompt template was refined through trial and error to align with the answer style present in `SQuAD v2`, so that the generated output is comparable to the ground-truth answers of the dataset.

This modular, object-oriented design allows the overall system to be altered by replacing components, without modifying core logic. This is possible thanks to the proper use of design patterns[1]. On one hand, the *Strategy* pattern allows two interchangeable backends in the indexing and retrieval: an in-memory and a persistent Chroma store, or a web-based or text-based loader. This interchangeable pattern not only allows selecting one or the other without altering the pipeline, but also allows new additional components to be added, making the architecture scalable. On the other hand, the *Factory* pattern allows a dedicated factory to instantiate a chat model provider, so that the system can run a locally served model or a hosted provider.

[1]Design patterns are general, reusable solutions to recurring problems in software design. Rather than finished pieces of code, they are templates describing how to structure classes and objects to make a system more flexible, modular and maintainable (Gamma et al., 1994).

Optimization and query-aware layer

On top of the RAG system, the optimization layer searches the action space and a query-aware layer groups questions so that the configuration can be particular to the group, rather than global. Together, they implement the offline phase and serve to feed the online phase, as can be seen in Figure 6.

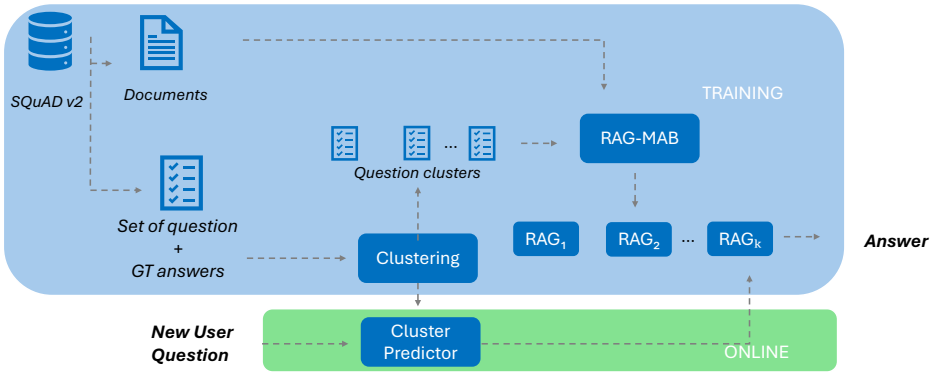


Figure 6: Training and deployment pipeline of the system.

The interaction between the RAG system and the optimizer is shown in Figure 7: given a question from the dataset, the RAG system produces an answer, the rewarder scores it against the ground-truth answer using Equation (3.1), and the signal is fed back to the bandit agent, which selects the next configuration to evaluate. Instead of evaluating every configuration through brute force, the search is driven by UCB, described in Algorithm 1, effectively balancing exploration and exploitation and focusing evaluation on the most promising actions. The result of this process is an estimate of the expected reward of each possible action, which can be visualized as a heatmap that maps the two-dimensional action space of chunk size and temperature into the estimated expected reward.

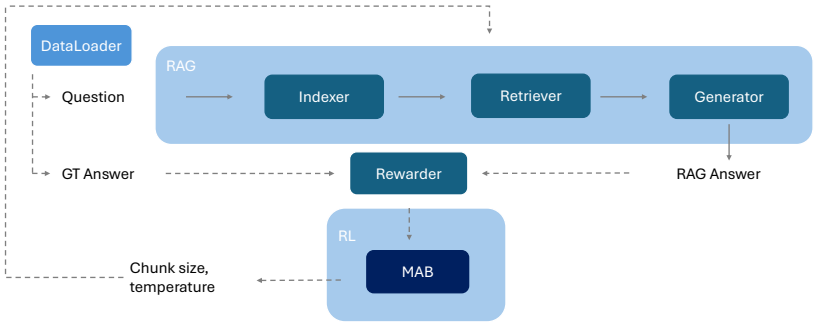


Figure 7: Architecture of the RAG system with bandit-based optimization.

Clustering of the questions. Once the questions are embedded into semantic embedding space, the embeddings are partitioned into clusters through hierarchical agglomerative clus-

tering using Ward linkage. This algorithm is used because it sets the appropriate number of clusters automatically by maximizing the silhouette score.

Per-cluster selection. Within each cluster, the bandit is run independently on every question, producing one reward heatmap per question. These heatmaps are then averaged, and the configuration assigned to the cluster is the arm with the highest average reward, recovering the cluster optimum a_k^* of Equation (3.4). Averaging across the questions of the cluster prevents noise and yields a configuration that is representative of the cluster as a whole.

Cluster prediction. After training, during evaluation, previously unseen questions must be assigned to a cluster of questions whose configuration has been optimized in advance during training. This is done by a weighted k-nearest neighbors predictor that operates on the same embedding space; it selects the k nearest neighbors weighted by their cosine similarity, and the question is then assigned to the cluster with the highest score. The configuration a_k^* learned for that cluster is then applied to answer the question.

Technical justification. The use of a multi-armed bandit, rather than a full reinforcement learning approach, such as policy-gradient optimization, is appropriate because the system is choosing amongst a small, fixed set of discrete actions and observing the reward of each choice. The reward of an action does not depend on any sequential state, since there are no state transitions and therefore no long-term cumulative reward. In this setting, UCB is a more suitable arm-selection strategy because it balances exploration and exploitation in a simple stationary bandit problem. In terms of clustering, hierarchical agglomerative clustering was used to automatically select the number of clusters and does not require committing to a fixed number of clusters.

3.4 Implementation

This section describes how the design presented above is implemented: the organization of the software into modules, executable pipelines that connect online and training phases, and the measures taken to ensure reproducibility.

Code structure

The system is implemented as a Python library, `pluto-mab`, divided into independent modules responsible for a specific part of the system.

The layout of the structure is presented below:

```
pluto/
├── rag/                      RAG system
│   ├── rag.py                  abstract components + RAG class
│   ├── rag_interface.py        high-level interface (InMemory / Chroma)
│   ├── config.py               RAG configuration object
│   ├── llm_loader.py           language-model factory
│   └── utils.py                prompt and similarity helpers
├── rl/                       action space and reward
│   └── rl.py                   RagAction, Rewarder, DatasetRewardScorer
├── clustering/               query-aware layer
│   ├── clusters.py             agglomerative question clusterer
│   ├── cluster_predictor.py    weighted k-NN cluster predictor
│   ├── embeders.py             embedding utilities
│   └── viz.py                  clustering plots
├── selection/                per-cluster configuration learning
│   ├── config_selector.py      bandit-based selector
│   ├── config_predictor.py     configuration predictor
│   └── selection_pipeline.py   selection orchestration
├── eval/                     evaluation
│   ├── evaluators.py           bandit (UCB) and brute-force search
│   ├── experiment.py           experiment runner
│   ├── datasets.py             SQuAD v2 loader
│   └── viz.py                  result figures
└── pipelines/                end-to-end orchestration
    ├── training_pipeline.py    offline stage
    └── online_pipeline.py      online stage
```

This structure shows the code is organized in six independent modules: `rag` holds the Retrieval-Augmented Generation system; `rl` defines the reinforcement learning formulations: the action representation (`RagAction` built from a chunk-size and temperature pair) and the reward (a `Rewarder` base class from which various rewarders can inherit: `DatasetRewardScorer` uses cosine similarity); `clustering` and `selection` implement the query-aware layer, clustering questions and learning a cluster-specific configuration; `eval` contains the bandit and brute-force strategies, the experiment runner and the dataset loader as well as plotting utilities; finally, `pipelines` orchestrates the training and online stages.

In addition, the repository includes auxiliary directories for scripts, configuration files, artifacts, and tests, which support experimentation but are beyond the core system.

Training and online pipelines

The offline stage, in `training_pipeline.py`, takes a document along with its questions and ground-truth answers, builds the action space and a base configuration, and runs the query-aware optimization algorithm iteratively. Then, its output is a `TrainingSelectionArtifact`: a serialized object encapsulating the per-cluster configurations and the clustered training

questions.

The online stage, in `online_pipeline.py`, is run by the `OnlineRAGPipeline` class; it loads the artifact from the training pipeline and sets up the k-nearest-neighbors predictor from the stored questions. Afterwards, for each new unseen question, it predicts the cluster the question belongs to and generates the answer with the RAG whose configuration has been learned for that cluster.

Reproducibility

Several measures are taken to support the reproducibility of the system. At the beginning of the training stage, the seeds of the two random number generators, `random` and `NumPy`, are fixed from a single `seed` value. This makes the sampling of the questions repeatable across runs. Also, all the configurable parameters of a run, such as the number of trials, or the number of questions, are exposed as command line arguments. In addition, the default runtime configurations, such as model names and base RAG hyperparameters, are defined in a `default.yaml` file. This allows the experimental setup to be explicit and repeatable.

Finally, the trained system is serialized into a single artifact and all dependencies and requirements are compacted in a `pyproject.toml`, so that the project can be easily reconstructed on another machine.

4 EXPERIMENTAL RESULTS

The experiments were run on the materials and parameters already described. They used a single random SQuAD v2 article, from which 60 answerable questions are sampled. The bandit runs 100 trials and stores intermediate snapshots every 25 trials. Exhaustive brute-force evaluates all 70 configurations 5 times, averaging the resulting rewards to reduce noise introduced by language model generation. All runs have a fixed seed to ensure reproducibility. The static baseline used for comparison is a fixed configuration with a chunk size of 200, a chunk overlap of 40 and a temperature of 0.3.

In terms of the evaluation metrics, three quantities are reported. First, the similarity between the rewards estimated by the bandit and those obtained through brute-force evaluation is measured. Since brute-force search evaluates every configuration exhaustively, its reward estimates are taken as the reference estimate: the mean squared error (MSE) and the difference between the two heatmaps reveal how accurately the multi-armed bandit builds the reward landscape. Second, the computational efficiency of the multi-armed bandit is measured in terms of time, in order to justify its use against brute-force evaluation. Finally, the total accumulated reward of the query-aware selector is compared against that obtained by a static baseline RAG, providing evidence for the usefulness of the overall system.

The optimization is performed over the 70 possible configurations of the action space defined in Section 3.2. All remaining RAG hyperparameters are kept fixed during the evaluation, including a retrieval depth of `top_k` = 4, as well as the embedding and language models described in Section 3.

The components shared by the three methods and the hyperparameters optimized by each are summarized in Table 1 and Table 2, respectively.

Parameter	**Value**
Language model	Llama 3.1 8B (Ollama)
Embedding model	`all-mpnet-base-v2`
Retrieved passages (`top_k`)	4
LLM `top_k`	10
Context window (`num_ctx`)	4096
Prompt	Extractive (`SHORT_ANSWER_PROMPT`)

Table 1: Fixed RAG components shared by the three methods.

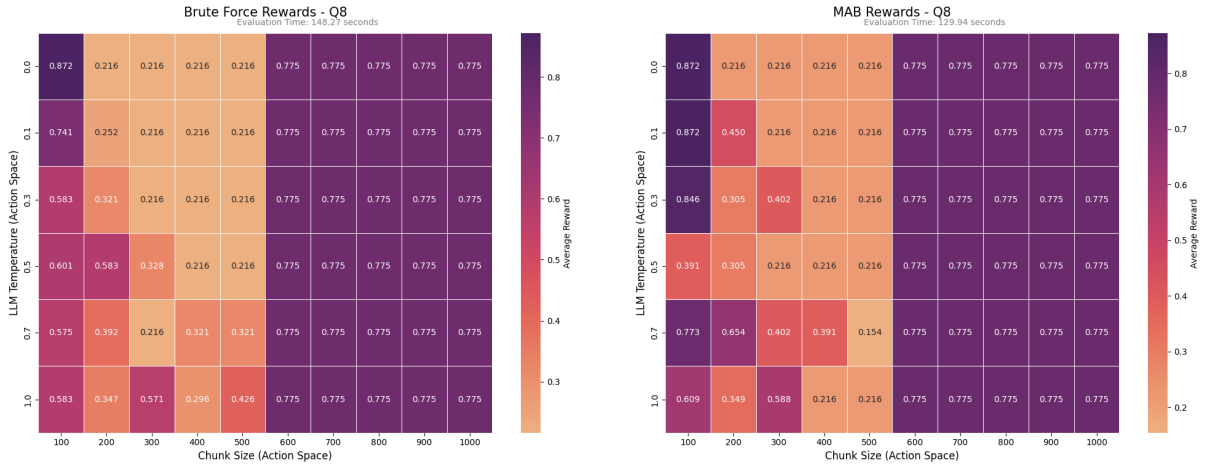
Method	Chunk size	Chunk overlap	Temperature
Baseline	200	40	0.3
MAB-flat	from artifact	chunk size/5	from artifact
MAB-clustered	per cluster (artifact)	chunk size/5	per cluster (artifact)

Table 2: Optimized hyperparameters for each method.

4.1 MAB versus Brute Force

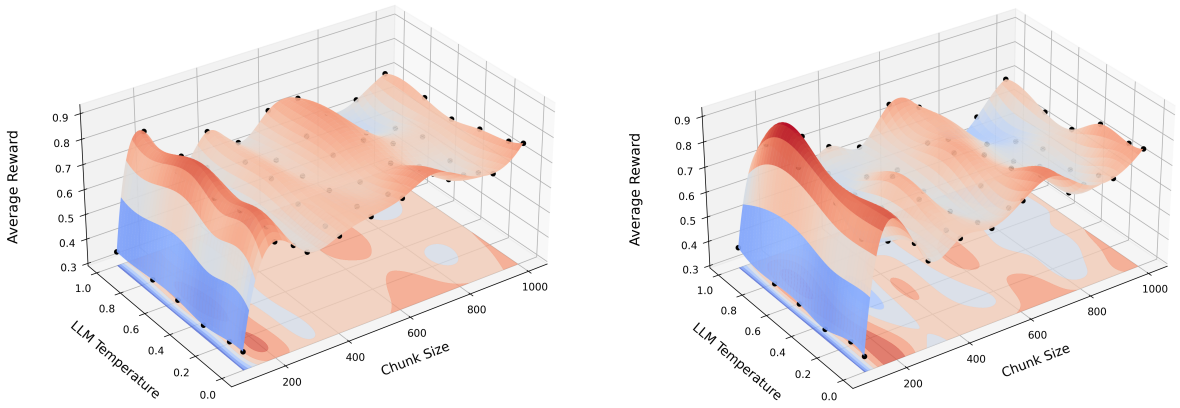
The first set of experiments assesses whether multi-armed bandit is able to recover the same reward landscape as an exhaustive brute-force search.

Reward landscape for a single question. Figure 8 shows the estimated reward heatmap over chunk size (horizontal axis) and temperature (vertical axis) for a single question; obtained by brute force (a) and multi-armed bandit (b). It can be seen that the difference between these two heatmaps is insignificant; both exhibit similar reward landscape with the same high and low density areas. This is shown in Figure 9 where discrete rewards values are interpolated into a three-dimensional surface. This indicates that the bandit identifies the same promising regions as exhaustive search.



(a) Brute Force (b) Multi-Armed Bandit

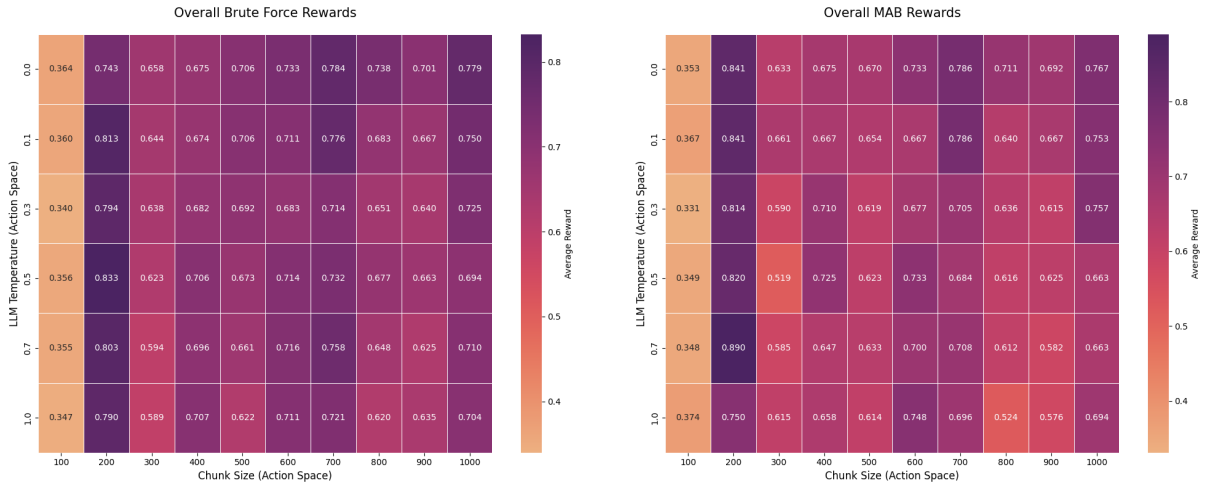
Figure 8: Estimated reward over the action space (chunk size × temperature) obtained by brute force and by the bandit.



(a) Brute Force (b) Multi-Armed Bandit

Figure 9: Interpolated reward surface over the action space (chunk size × temperature) obtained by brute force and by the bandit.

Aggregated reward landscape. To ensure a consistent comparison, the same evaluation is repeated over a batch of questions and the resulting results are averaged for each configuration. Figure 10 shows the mean reward landscape over 20 questions, obtained with brute force (a) and with the bandit (b). Similar to the single-question analysis, both heatmaps exhibit reward patterns along action space, supporting the reliability of a bandit as an efficient alternative to brute-force search.



(a) Brute Force (average)

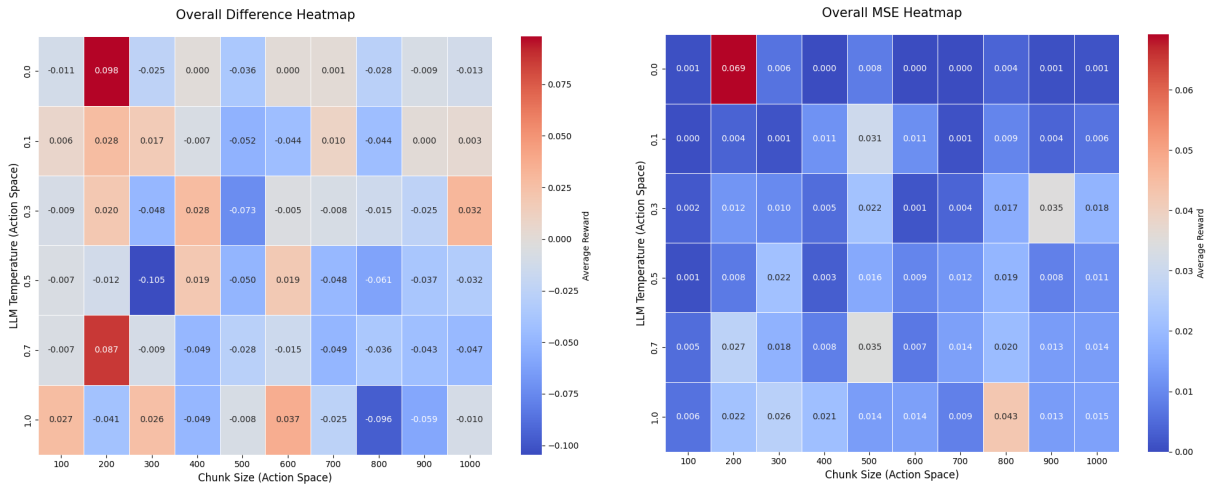
(b) Multi-Armed Bandit (average)

Figure 10: Mean estimated reward over the action space, averaged across 20 questions, obtained by brute force and by the bandit.

Quantifying the discrepancy. In order to further quantify the correspondence between brute force and bandits beyond visual inspection, the discrepancy between the two methods is computed, for each configuration $a = (s, \tau)$, and each question q_i , and then aggregated over a batch of 20 questions. Two quantities are reported: the mean difference and the mean squared error:

$$\Delta(a) = \frac{1}{N}\sum_{i=1}^{N}\big(r_{\mathrm{MAB}}(a;q_i) - r_{\mathrm{BF}}(a;q_i)\big), \qquad \mathrm{MSE}(a) = \frac{1}{N}\sum_{i=1}^{N}\big(r_{\mathrm{BF}}(a;q_i) - r_{\mathrm{MAB}}(a;q_i)\big)^2, \tag{4.1}$$

where $r_{\mathrm{BF}}(a;q_i)$ and $r_{\mathrm{MAB}}(a;q_i)$ denote the reward estimated for configuration a on question q_i by brute force and by the MAB, respectively, and $N = 20$ is the batch size. The difference indicates whether the bandit systematically over or under estimates the reward of a configuration, while the squared error captures the magnitude of the deviation regardless of its sign. Both are shown in Figure 11.



(a) $\Delta(a) = r_{\text{MAB}}(a) - r_{\text{BF}}(a)$ (b) $\text{MSE}(a) = \big(r_{\text{BF}}(a) - r_{\text{MAB}}(a)\big)^2$

Figure 11: Discrepancy between brute force and bandit over the action space (chunk size $\times$ temperature): (a) Difference and (b) Mean Squared Error.

The differences remain insignificant and around zero over all action space. This confirms that the bandit reconstructs the brute-force accurately, without any configuration deviating significantly.

Convergence of the bandit. The estimates reported above are learned over a series of trials. Figure 12 shows snapshots of the bandit at trials 25, 50, 75 and 100 for a single question. The top row exhibits the current estimated reward of each configuration at that trial. The bottom row shows the number of times each configuration has been pulled (counts). Each column represents a number of trials, which increases as you move through the columns. In the reward row (top row), many configurations have not been explored and thus appear at zero; as the search advances, they are gradually filled in, and by trial 100 the estimated landscape matches the brute-force one. In the meantime, the count row reveals the underlying behavior of UCB (Auer et al., 2002): each configuration is first played once (trials 25 and 50), and once all configurations have been explored, the remaining trials are allocated to those with the highest rewards. Hence, Figure 12 shows how UCB gradually shifts from exploration towards greedy exploitation. It is important to notice that the trial budget (100) is larger than the action space (70 configurations); this means that the initial steps are spent on a sweep across the action space, while the remaining steps focus on the most promising rewards.

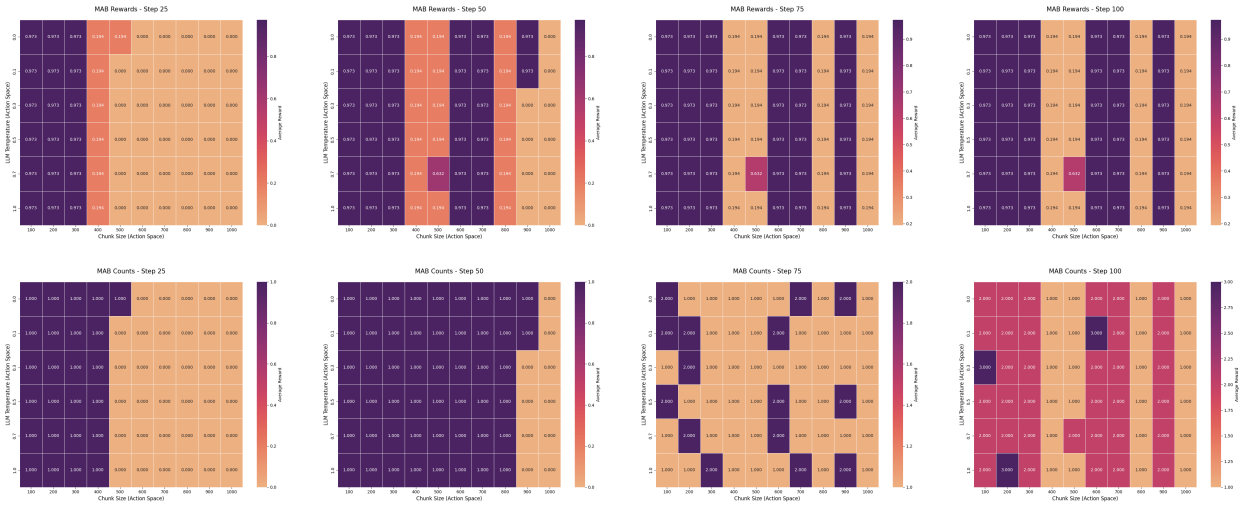


Figure 12: Evolution of the bandit across trials (25, 50, 75, 100, from left to right) for a representative question: estimated reward per configuration (top row) and number of pulls per configuration (bottom row).

Computational efficiency. In addition to approximating the reward landscape obtained by brute-force, the MAB approach provides a computational advantage. Figure 13 compares the evaluation times of both methods across the first 10 questions of the heatmaps above, showing that MAB consistently requires less time; reducing average evaluation time from 205.68 s to 78.10 s.

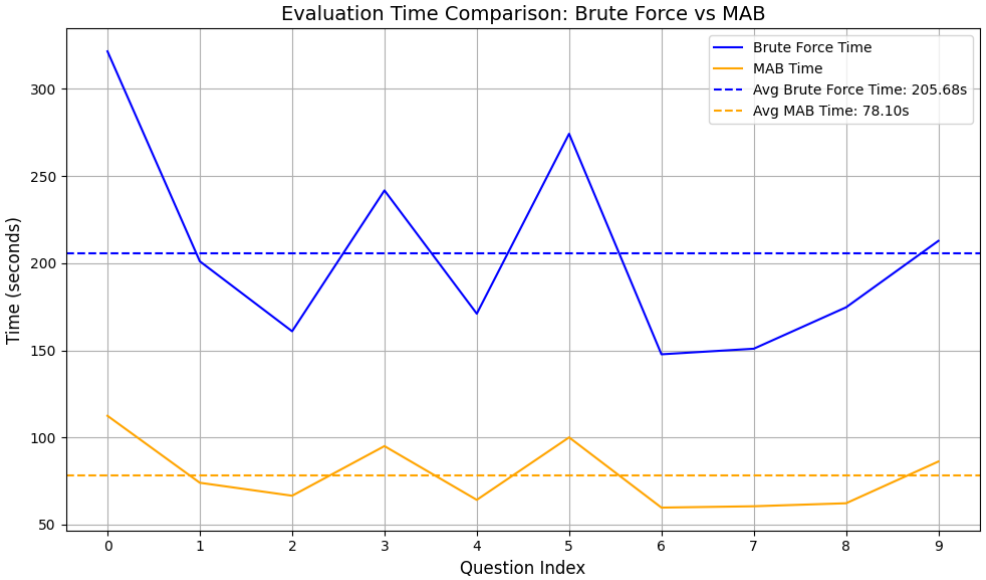


Figure 13: Evaluation time per question for brute force and the bandit, with their respective averages.

This indicates that the proposed approach is not only able to reproduce the same rewards pattern, but that also does it in a substantially more efficient way.

The reason for this difference is not that bandit performs operations in a more efficient way as both configurations have the same single-language model call and $O(1)$ operations of comparable cost. However, the *number* of calls each method requires changes. Brute force must evaluate all 70 configurations and, in order to reduce the noise introduced by stochastic generation, each evaluation is repeated 5 times; resulting in $70 \times 5 = 350$ language-model calls per question. In contrast, the MAB is evaluated only for 100 trials per question, and thus makes 100 calls. Since the language-model call is the most expensive part of the pipeline, the reduction from 350 to 100 calls results in a significant difference when it comes to evaluation time. This is summarized in Table 3.

Method	**RAG calls per question**	**Mean time per question**
Brute force	$70 \times 5 = 350$	205.68 s
Multi-Armed Bandit	100	78.10 s

Table 3: Evaluation cost comparison between brute force and MAB.

4.2 Query-aware selection

The second part of the experiments consists of evaluating the complete system against a static RAG baseline configuration. In order to highlight the contribution of clustering, three conditions were compared: full query-aware system (*MAB-clustered*), which represents the whole pipeline; a simplified version (*MAB-flat*), which uses a single bandit to learn one configuration for all the questions, without clustering; and a *Baseline RAG* with a fixed configuration. All three are evaluated in the same article over 20 trials, with the same hyperparameters, excluding those optimized by MAB, to ensure consistency.

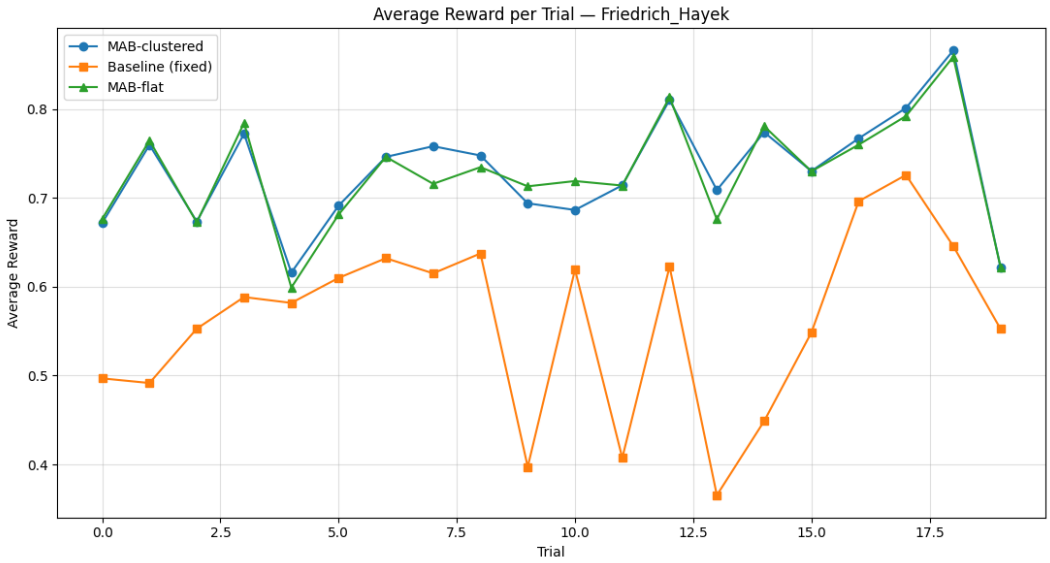


Figure 14: Average reward per trial for the MAB-clustered, MAB-flat and the static baseline, on the `Friedrich_Hayek` article.

Figure 14 shows the average reward per trial of the three conditions. Table 4 summarizes their statistics. The table shows that the difference between MAB methods and the baseline is not only significant, but consistent across trials. MAB-clustered averages a reward of 0.7305 and MAB-flat of 0.7276, while baseline averages 0.5618. Also, baseline never achieves the best result at any trial (0% wins), whereas both MAB methods, clustered and flat, are best in 60% and 40% of the trials respectively. In addition, the baseline exhibits a significantly higher variance (0.0969 versus around 0.06), indicating that a single fixed configuration performs unevenly across questions.

Method	**Mean**	**Std**	**Min**	**Max**	**Wins**
MAB-clustered	0.7305	0.0608	0.6159	0.8659	60%
MAB-flat	0.7276	0.0618	0.5990	0.8587	40%
Baseline (fixed)	0.5618	0.0969	0.3654	0.7258	0%

Table 4: Average-reward statistics over 20 trials for the MAB-clustered, MAB-flat and the static baseline on the `Friedrich_Hayek` article.

Overall, these results suggest that the main performance gain comes from adapting RAG to a suitable configuration, rather than relying on a fixed configuration throughout the evaluation. Both MAB methods outperform the static baseline in a consistent way. In the meantime, little difference is reported between the two MAB methods, MAB-clustered and MAB-flat, suggesting that most of the improvement comes from the bandit method itself.

Although clustering provides additional benefit, as reflected in the win rate of MAB-clustered against MAB-flat, its effect on the reward still remains small.

The small gain of clustering over the MAB-flat can be explained by the geometry of the questions themselves. For the `Friedrich_Hayek` article, MAB-clustered partitions the 60 questions into two clusters. Figure 15 shows a two dimensional projection of the 768-dimensional question embeddings, colored by cluster, obtained with Principal Component Analysis (PCA) [2], apparently without a clear boundary. As shown at the top of the figure, the first principal component explains 12.5% of the variance and the second 8.3%, adding up to only 20.8% of the total variance. Thus, only seeing almost a fifth part of real structure and hence clusters might look closer or separated in original 768-dimensional space. That's why silhouette score is a more reliable metric as it is computed over the full dimension. In this case, global silhouette score is low (0.168), confirming that the clusters overlap.

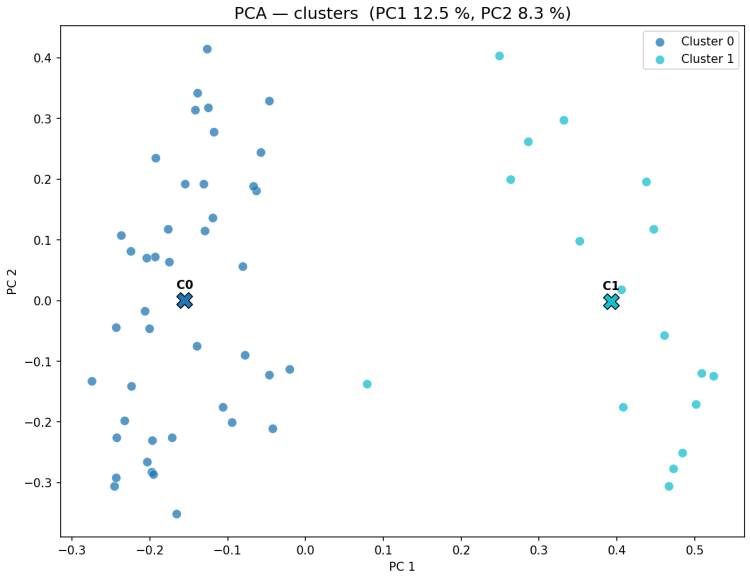


Figure 15: Two-dimensional PCA projection of the question embeddings of the `Friedrich_Hayek` article, colored by cluster.

Even though clusters are weakly separated, they still capture an interpretable distinction. Table 5 reports, for each cluster: its size, its selected configuration by *MAB-clustered* and its quality. The clusters clearly select very distinct configurations: the larger cluster favors a large chunk size and non-zero temperature, while the smaller cluster favors a smaller chunk size and a deterministic zero-temperature. Questions grouped by clusters are exhibited in Appendix C. The clustering captures a partition that improves the reward slightly, but that does not correspond to a clearly interpretable distinction.

[2]Principal Component Analysis is a linear dimensionality-reduction technique that projects high-dimensional data onto the directions of greatest variance, allowing the 768-dimensional question embeddings to be visualized in two dimensions while preserving as much of their variability as possible (Jolliffe, 2002).

Cluster	Questions	Chunk size	Temperature	Best score	Silhouette
0	43 (71.7%)	800	0.2	0.7748	0.247
1	17 (28.3%)	400	0.0	0.9378	−0.031

Table 5: Clusters found for the `Friedrich_Hayek` article: size, configuration selected by the per-cluster bandit, best reward, and mean silhouette score per cluster.

Further analysis of clusters found for the `Friedrich_Hayek` article is presented in Appendix D.

4.3 Critical Analysis and Limitations

Although the results above support the two claims of this work, several limitations are to be considered.

First, the rewards are computed between the ground-truth answer and the generated answer, which captures semantic closeness, but not necessarily factual correctness; an answer may be rewarded for being semantically similar while not being factually right.

Second, the clustering contribution could not be fully demonstrated. As shown above, the questions of a single article are semantically homogeneous and therefore cannot be clearly separated into clusters, so a single configuration already captures most of the benefit. Consequently, the clustering mechanism is expected to be used on more heterogeneous questions.

Third, for the sake of this work, the evaluation is restricted in scope. The system is assessed on individual SQuAD v2 articles using a locally served language model. The results should be presented as a proof of concept, rather than a large scale benchmark.

Finally, optimization is affected by the stochastic nature of LLMs and the size of the trial budget. Generation with a non-zero temperature, although mitigated by averaging, makes a noisy reward signal. Furthermore, with 70 configurations and only 100 trials, a large part of the budget is spent on evaluating each configuration once. Hence, the real efficiency of using bandits comes in larger action spaces where brute force would simply become impractical.

5 CONCLUSIONS AND FUTURE WORK

5.1 Conclusions

This work set out to study whether or not the hyperparameter configurations of a RAG system can be automated and optimized more efficiently than by exhaustive brute-force search, while also adapting to the questions being asked. Given the results presented in Section 4, all four objectives presented in Section 1.2 have been fulfilled.

First, the process of configuring RAG hyperparameters has been fully automated. The system is able to automatically select the chunk size and temperature for a given document and a set of questions and ground-truth answers, without any human intervention; thus reducing the manual trial-and-error that RAG systems currently rely on.

Second, the multi-armed bandit has proven to be more efficient than brute-force search. As seen in the results, the UCB algorithm can reconstruct the brute-force reward landscape in a significantly smaller amount of time, due to the lower number of model calls per question. The bandit not only approximates the exhaustive search results, but also does it at almost a third of the computational cost. It is important to note that this would be especially convenient when increasing the size of the action space, where brute force would simply become impractical.

Third, the adaptive selection through the clustering mechanism consistently outperforms a static baseline configuration. Both MAB variants achieve a considerably higher mean reward than the fixed baseline average. Also, the baseline exhibits a higher variance, confirming that a single fixed configuration performs unevenly across different questions.

Finally, the proposed evaluation framework, based on cosine similarity on pre-defined ground-truth answers, provided a consistent and reproducible criterion, allowing the different methods to be evaluated under the same conditions.

Altogether, even though the objectives were achieved, the contribution of the clustering could not be fully demonstrated and remains limited. It was shown above that the gains of MAB-clustered over MAB-flat were small, and it was justified by the analysis of the embedding-space clusters; showing semantically homogeneous questions, with little structure to allow per-cluster bandit exploitation. Yet, clustering was able to select distinct configurations, indicating that this architecture performed as expected and perhaps it may benefit from more heterogeneous question sets.

5.2 Future Work

In light of the above, several lines of future work naturally emerge from this project.

Heterogeneous domains. The clustering mechanism shall be validated in a setting with a set of heterogeneous questions that exhibit clearly differentiated patterns. For instance, in a customer-service call center, questions naturally split into recognizable groups, each of which may benefit from a different configuration.

Broader action spaces. This work formulated the problem with a two-dimensional action space for the temperature and the chunk size. However, this formulation can be extended to different hyperparameters and even more dimensions, such as top-k, chunk overlap or the embedding model. Since the cost of an exhaustive search grows multiplicatively with the number of dimensions, the efficiency of bandits becomes particularly relevant in this setting.

More sophisticated RAG architectures. This work was designed to be scalable and is not tied to a particular RAG pipeline, and thus can be integrated with more sophisticated pipelines involving re-ranking techniques, hybrid retrieval or query rewriting, where manual tuning becomes impractical.

From bandits to full reinforcement learning. Looking further ahead, the bandit formulation could be extended to a full reinforcement learning setting with states encoding the conversational context, allowing a configuration policy to adapt not only to the questions, but also to the history of the dialogue.

6 REFERENCES

[1] Agnihotram, G., Sarkar, J., & Kasthuri, M. (2025). Beyond static retrieval: A reinforcement learning framework for dynamic and adaptive rag. *American Journal of Computer Science and Technology*, *8*(4), 181–188. https://doi.org/10.11648/j.ajcst.20250804.12

[2] Auer, P., Cesa-Bianchi, N., & Fischer, P. (2002). Finite-time analysis of the multiarmed bandit problem. *Machine Learning*, *47*(2–3), 235–256. https://doi.org/10.1023/A:1013689704352

[3] BBVA AI Factory. (2024). *Our rag-based information agent* [Accessed: 2026-05-12]. https://www.bbvaaifactory.com/our-rag-based-information-agent/

[4] Fu, J., Qin, X., Yang, F., Wang, L., Zhang, J., Lin, Q., Chen, Y., Zhang, D., Rajmohan, S., & Zhang, Q. (2024). Autorag-hp: Automatic online hyper-parameter tuning for retrieval-augmented generation. *Findings of the Association for Computational Linguistics: EMNLP 2024*, 3875–3891.

[5] Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design patterns: Elements of reusable object-oriented software*. Addison-Wesley.

[6] Google Cloud. (2024). *Optimizing rag retrieval: Test, tune, succeed* [Accessed: 2026-05-12]. https://cloud.google.com/blog/products/ai-machine-learning/optimizing-rag-retrieval

[7] Huang, J., Madala, S., Sidhu, R., Niu, C., Peng, H., Hockenmaier, J., & Zhang, T. (2025). Rag-rl: Advancing retrieval-augmented generation via rl and curriculum learning. *arXiv preprint arXiv:2503.12759*.

[8] James, G., Witten, D., Hastie, T., & Tibshirani, R. (2021). *An introduction to statistical learning: With applications in r* (2nd ed.). Springer. https://doi.org/10.1007/978-1-0716-1418-1

[9] Jolliffe, I. T. (2002). *Principal component analysis* (2nd). Springer.

[10] Lewis, P., Perez, E., Piktus, A., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in Neural Information Processing Systems*.

[11] Llama Team, AI @ Meta. (2024). The Llama 3 herd of models. *arXiv preprint arXiv:2407.21783*. https://doi.org/10.48550/arXiv.2407.21783

[12] MIT News. (2025). *Explained: Generative ai's environmental impact* [Accessed: 2026-05-12]. https://news.mit.edu/2025/explained-generative-ai-environmental-impact-0117

[13] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., & Duchesnay, É. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*, 2825–2830.

[14] Rajpurkar, P., Jia, R., & Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 784–789. https://doi.org/10.18653/v1/P18-2124

[15] Reimers, N., & Gurevych, I. (2019). Sentence-BERT: Sentence embeddings using siamese BERT-networks. *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 3982–3992. https://doi.org/10.18653/v1/D19-1410

[16] Song, K., Tan, X., Qin, T., Lu, J., & Liu, T.-Y. (2020). MPNet: Masked and permuted pre-training for language understanding. *Advances in Neural Information Processing Systems (NeurIPS)*, *33*, 16857–16867.

[17] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.

[18] United Nations. (2024). *Sustainable development goals* [Accessed: 2026-05-12]. https://sdgs.un.org/goals

A Source Code

The complete implementation of the system described in this work is openly available at:

github.com/Ibinarriaga8/pluto

It is published on the Python Package Index (PyPI) under the name `pluto-mab` and can be installed in any environment with a single command:

```
pip install pluto-mab
```

A minimal example reproducing the query-aware selection procedure is shown in Listing 1; the complete library usage manual is documented in the repository:

```python
from pluto.eval.datasets import SquadV2Loader
from pluto.rag.config import RAGConfig
from pluto.rag.utils import SHORT_ANSWER_PROMPT
from pluto.rl.rl import DatasetRewardScorer, RagAction
from pluto.clustering.embeders import QuestionEmbedder
from pluto.clustering.clusters import AgglomerativeQuestionClusterer
from pluto.selection.config_selector import MABSelector
from pluto.selection.selection_pipeline import ClusterSelectionPipeline

# Load a random SQuAD v2 article
loader = SquadV2Loader()
title, items, long_context = loader.load_random_article()
print(f"Article: {title} ({len(items)} questions)")

# RAG config with the article as context
config = RAGConfig(
    texts=[long_context],
    llm_model_name="llama3.1:8b",
    llm_provider="ollama",
    custom_prompt_template=SHORT_ANSWER_PROMPT,
)
action_space = [
    RagAction(chunk_size=k, llm_temperature=t)
    for k in [100, 300, 500, 700, 1000]
    for t in [0.0, 0.3, 0.7, 1.0]
]

# UCB bandit selector
selector = MABSelector(
    config_template=config,
    reward_scorer=DatasetRewardScorer(),
    action_space=action_space,
)

# Cluster questions and optimize one configuration per cluster
embedder = QuestionEmbedder("sentence-transformers/all-mpnet-base-v2")
clusterer = AgglomerativeQuestionClusterer(embedder=embedder)
pipeline = ClusterSelectionPipeline(clusterer=clusterer, selector=selector)
result = pipeline.run(items=items, min_clusters=2, max_clusters=5, trials=100)
```

Algorithm 1: Minimal usage of `pluto-mab`: query-aware configuration selection.

B Reference Implementation and Illustrative Behavior of the UCB1 Policy

This appendix provides a Python implementation to illustrate the UCB1 algorithm (Algorithm 1).

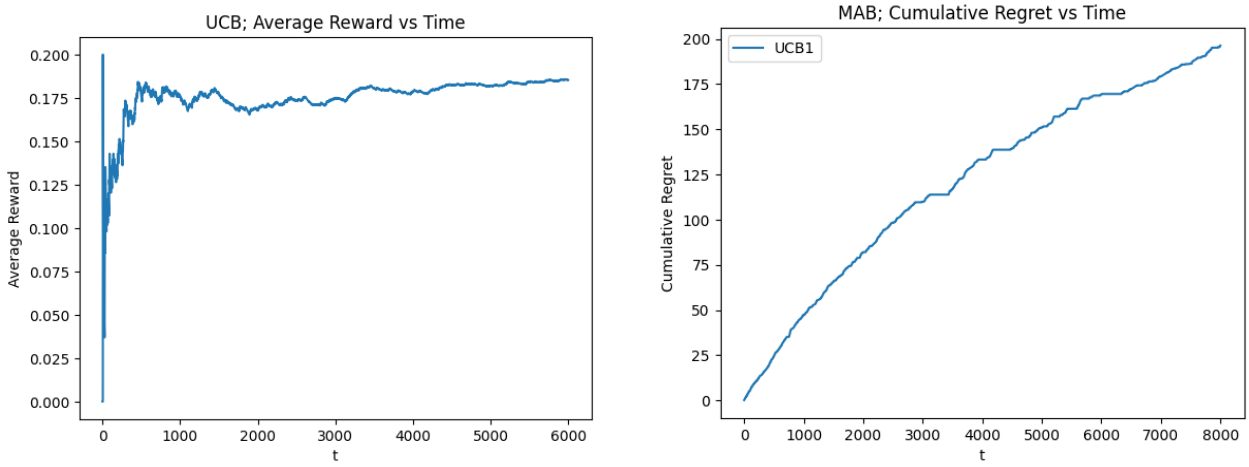
The code implements Algorithm 1 directly in a simplified Bernoulli bandit setting. `ToyBandit` defines the arms and generates rewards, while `ToyUCBEvaluator` keeps track of $N_a(t)$ through `counts` and $\hat{\mu}_a(t)$ through `values`. The method `_select_arm` first explores the environment by pulling each arm at least once and then selects the arm with the highest UCB score, combining the empirical mean with an exploration bonus. After each pull, `_update_arm` increments the selected arm's count and updates its empirical mean using the same incremental update rule as UCB1.

This toy example follows the algorithm step by step and is later extrapolated to the full system, replacing the stationary Bernoulli bandit with the full RAG configuration space.

```python
import matplotlib.pyplot as plt
import numpy as np


class ToyBandit:
    """
    K-armed bandit with fixed (unknown) reward means in [0, 1].
    Represents the environment for online hyper-parameter tuning.
    """
    def __init__(self, probs, seed=0):
        self.probs = np.array(probs, dtype=float)
        self.K = len(probs)
        self.rng = np.random.default_rng(seed)

    def pull(self, arm: int) -> float:
        # Bernoulli reward in {0, 1}, subset of [0, 1]
        # sample from arm distribution
        return float(self.rng.random() < self.probs[arm])

    @property
    def best_mean(self):
        return float(np.max(self.probs))


class ToyUCBEvaluator:
    """
    Toy implementation of UCB MAB.
    Mirrors the structure of the earlier RAGUCBEvaluator.
    """
    def __init__(self, bandit: ToyBandit, alpha: float = 1.0, seed=0):
        self.bandit = bandit
        self.alpha = float(alpha)  # alpha controls exploration vs exploitation
        self.rng = np.random.default_rng(seed)

        self.K = bandit.K
        self.t = 0
        self.counts = np.zeros(self.K, dtype=np.int64)
        self.values = np.zeros(self.K, dtype=np.float64)

    def _select_arm(self) -> int:
        """
        UCB arm selection.
```

```
        """
        # Pull each arm once
        for a in range(self.K):
            if self.counts[a] == 0:
                return a

        t = self.t + 1
        bonus = self.alpha * np.sqrt(np.log(t) / self.counts)
        ucb = self.values + bonus

        return int(np.argmax(ucb))

    def _update_arm(self, arm: int, reward: float) -> None:
        """
        Incremental mean update.
        """
        # update expected reward for that arm
        self.counts[arm] += 1
        n = self.counts[arm]
        self.values[arm] += (reward - self.values[arm]) / n  # incremental mean update

    def evaluate(self, trials: int = 5000):
        """
        Runs the bandit loop and returns learning curves.
        """
        rewards = np.zeros(trials)
        avg_rewards = np.zeros(trials)
        regrets = np.zeros(trials)

        mu_star = self.bandit.best_mean

        for t in range(trials):

            arm = self._select_arm()        # select an arm
            reward = self.bandit.pull(arm)  # compute reward for that arm

            self._update_arm(arm, reward)   # update expected reward for that arm
            self.t += 1

            rewards[t] = reward
            avg_rewards[t] = rewards[: t + 1].mean()
            regrets[t] = (mu_star - self.bandit.probs[arm]) + (
                regrets[t - 1] if t > 0 else 0.0
            )

        return rewards, avg_rewards, regrets
```

Algorithm 2: Toy UCB1 evaluator on a stationary Bernoulli bandit.



(a) Running average reward

(b) Cumulative regret

Figure 16: Learning dynamics of the toy UCB1 evaluator for a single run on a stationary Bernoulli bandit.

(a) Running average reward as a function of the trial t: after an initial exploration phase, the curve stabilizes as it exploits the action space and the policy concentrates its pulls on the optimal arm.

(b) Cumulative regret over the same run: the sublinear, progressively flattening growth indicates that suboptimal arms are pulled increasingly rarely, consistent with the logarithmic regret bound of UCB1:

$$\mathbb{E}[R_T] = \mathcal{O}\left(\sum_{a\,:\,\Delta_a>0} \frac{\ln T}{\Delta_a}\right) \tag{B.1}$$

where $\Delta_a = \mu^* - \mu_a$ is the suboptimality gap of arm a (Auer et al., 2002).

C Question clusters of the `Friedrich_Hayek` article

This appendix lists the questions assigned to each cluster found for the `Friedrich_Hayek` article, together with the configuration selected for each cluster by the per-cluster bandit.

Cluster 0 — 43 questions (chunk size = 800, temperature = 0.2)

1. In opposition to conservatives, what group has Hayek's work influenced?
2. Hayek focused most of his economic works on the business cycle, money and what else?
3. Hayek's critical analysis of Keyne's work was published under what title?
4. Who was Hayek's father?
5. Which type of economy did Hayek believe the price mechanism to be less effective with?
6. Other than Max Weber, who was a notable influence to Hayek's statements regarding resource distribution?
7. Which of Popper's works was the first to grasp Hayek's attention?
8. Which of Hayek's works did Thatcher produce at the Conservative Research Department?
9. Whose work most notably influenced Hayek's argument regarding resource distribution?
10. At the end of the same day Hayek met with the Queen of England, what did he say?
11. Which of Hayek's books argued against the socialist price mechanism?
12. What work did Hayek begin in 1923?
13. One of Hayek's supporters served which US president?
14. What would Hayek's free-market not require in order to function properly?
15. What topic were Hayek's next two books going to cover?
16. Hayek believed the requirements for a socialist economy would lead to what?
17. Hayek believed that authoritarianism was very different from what?
18. When did Hayek begin presenting his ideas on the limits of human knowledge?
19. For whom did Hayek work upon being hired by Ludwig von Mises?
20. What did Hayek conclude regarding his brothers?
21. Who was August von Hayek's father?
22. Diamond states that the final result of Hayek's statements are what?
23. Which whom did Hayek share a Nobel prize?
24. Which fellow Vienna native was Hayek friends with?
25. How did Hayek wish to be referred to after his 1984 award?

26. What was the result of Hayek's Prices and Production?
27. The agreement Hayek criticized was between the British Labour government and which political party?
28. What is the responsibility of government in Hayek's market order?
29. What has Hayek's views on the market been used to defend?
30. What sort of system did Hayek propose the government create?
31. Which ideology did Hayek believe conservatism discouraged?
32. Who was particularly critical of Hayek's work following Prices and Production?
33. What was the name of the ideology Hayek criticized?
34. Who was it that claimed Hayek's The Constitution of Liberty to be an thorough example of neoliberal philosophy?
35. Who's occupation inspired Hayek when he was older?
36. According to Hayek, clothing, food and shelter should be provided to what extent?
37. With whom did Hayek share his 1974 award?
38. Who made the claim that Hayek was particularly determined regarding his beliefs on social insurance and a safety net?
39. How did Hayek feel regarding income distribution?
40. In which nation was Friedrich Hayek born?
41. Which of Hayek's works did Friedman once teach?
42. Which type of government was more favorable than others according to Hayek?
43. What group did Hayek form with three other people?

Cluster 1 — 17 questions (chunk size = 400, temperature = 0.0)

1. What is the name of the book Friedman released in 1980?
2. As of the release of his 1960 book, how long had it been since The Road to Serfdom was released?
3. In regards to economics, Lionel Robbins believe English-speaking academics had what?
4. What does conservatism have in common with classical liberalism?
5. In scientism, it is typically believed that explanations in science are what?
6. Upon leaving London, for what college did he choose to work?
7. When was the final volume of Law, Legislation and Liberty released?
8. Whose 1980 book mentions "informal" economics?
9. What was the name of the book Wittgenstein published in 1921?

10. What did some Liberal politicians claim the pact was meant to do?
11. Who was responsible for England's return to the use of gold as standard currency?
12. Who disagreed with David Steel's statements in 1978?
13. What is the term used to describe economists following Keynes school of thought?
14. When was the Road to Serfdom published?
15. What did Wittgenstein and Hayek do during the first world war?
16. Lionel Robbins came to head which school in 1929?
17. What did socialists believe equilibrium theory invalidated?

D Additional Clustering Analysis

This appendix contains further analysis of the clustering stage that is not part of the main evaluation of the project, but characterizes the geometric structure of the high-dimensional question embeddings, underlying the results covered in Section 4.2.

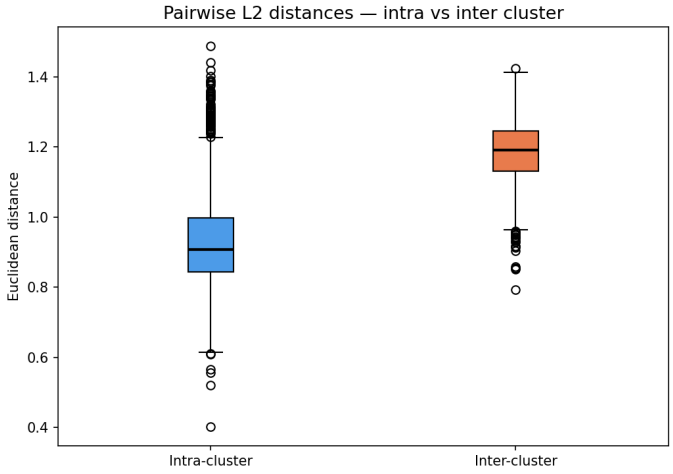


Figure 17: Distribution of pairwise Euclidean (L2) distances between question embeddings, separated into intra-cluster and inter-cluster pairs for the $K = 2$ partition.

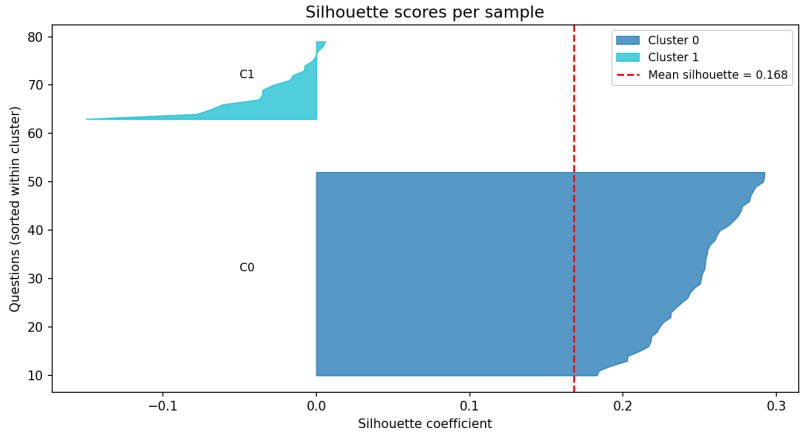


Figure 18: Silhouette coefficient of each question, sorted within cluster, for the $K = 2$ partition.

Figure 17 shows that inter-cluster pairs (questions from different groups) have a higher mean distance than intra-cluster pairs, confirming that the partition does capture some geometric structure. However, both distributions overlap considerably.

Figure 18, on the other hand, confirms this at the sample level: the mean silhouette of 0.168 and the negative coefficients in Cluster 1 indicate a weakly separated embedding space, consistent with the marginal gain of the clustered bandit over the flat variant in Section 4.2.