Instituto Tecnológico
y de Estudios Superiores de Occidente

Reconocimiento de validez oficial de estudios de nivel superior según acuerdo secretarial 15018, publicado en el Diario Oficial de la Federación del 29 de noviembre de 1976.

Departamento de Electrónica, Sistemas e Informática

Maestría en Sistemas Computacionales

Enhancing LLM performance in specialized Spanish domains using RAG and PEFT QLoRA

TRABAJO RECEPCIONAL que para obtener el **GRADO** de
MAESTRO EN SISTEMAS COMPUTACIONALES

Presenta: **ERICK BADILLO RANGEL**

Asesor: **DR. LUIS MIGUEL ESCOBAR VEGA**

Tlaquepaque, Jalisco. Noviembre 2024.

ACKNOWLEDGEMENTS

I want to thank my wife; her love, support, and companionship have been an inexhaustible motivation throughout the effort to complete this important milestone in my academic and professional life.

I am also grateful to my mother, father, and sister for their unconditional support, their words of encouragement, and the inspiring example they have always set through their actions.

I would like to thank ITESO and CONAHCYT (578347) for granting me the scholarships that allowed me to complete my master's program.

I am deeply thankful to the professors at ITESO for their invaluable mentorship and for sharing their knowledge and experiences. In particular, I thank Dr. Iván Villalón for his support in reviewing this work; his availability, clarity, and leadership are invaluable to the community of students in the Master's in Computer Systems program.

Lastly, I am grateful to Dr. Luis Miguel Escobar for his supervision during the development of this work.

AGRADECIMIENTOS

Agradezco a mi esposa, su amor, apoyo y compañía han sido una fuente inagotable de motivación a lo largo del esfuerzo realizado para concluir este importante hito en mi vida académica y profesional.

Agradezco a mi madre, padre y hermana por su apoyo incondicional, por sus palabras de aliento y por el inspirador ejemplo que siempre me han dado a través de su actuar en la vida.

Agradezco al ITESO y al CONAHCYT (578347) por haberme otorgado las becas que me permitieron completar mi programa de maestría.

Agradezco a los profesores del ITESO por su invaluable tutoría y por compartir sus conocimientos y experiencias; especialmente agradezco al Dr. Iván Villalón por su apoyo en la revisión de este trabajo, su disposición, claridad y liderazgo son invaluables para la comunidad de alumnos de la maestría en Sistemas Computacionales.

Finalmente, agradezco al Dr. Luis Miguel Escobar, por su supervisión durante la realización de este trabajo.

DEDICATION

I dedicate this work to my beloved daughters, Ximena and Natalia. Being your father is the greatest gift life has given me.

DEDICATORIA

Dedico este trabajo a mis amadas hijas, Ximena y Natalia. Ser su padre es el regalo más hermoso que me ha dado la vida.

ABSTRACT

This project proposes using RAG (Retrieval-Augmented Generation) combined with PEFT (Parameter-Efficient Fine-Tuning) and QLoRA (Quantized Low-Rank Adaptation) to enhance the performance of LLMs (large language models) in specialized knowledge domains in Spanish.

This work employs a large language model (LLM) to run four experiments, each evaluating the model's performance through a zero-shot approach on a set of instructions classified into open-ended, closed-ended, and summarization questions. The first experiment establishes a baseline for performance by conducting inferences in a vanilla configuration. The second experiment extends this by incorporating a retrieval-augmented generation (RAG) module. The third and fourth experiments utilize the best-performing model obtained through fine-tuning with the PEFT QLoRA technique, with the difference being that the fourth experiment integrates a RAG module.

The data used for RAG and fine-tuning was synthetically created through an ETL process deployed in the cloud, implemented in a serverless architecture, and designed following the data enrichment principles proposed by the medallion architecture. The ETL process enables scalability, reliability, and a high level of maintainability. For validation and experimentation purposes, only the "Ley de Impuesto sobre la Renta 2024" text body was used to execute the experiments described in this work.

The evaluation metrics used in this work include BERTScore (Bidirectional Encoder Representations from Transformers Score), ROUGE (Recall-Oriented Understudy for Gisting Evaluation), and BLEU (Bilingual Evaluation Understudy). BERTScore captures semantic similarity, while ROUGE assesses the overlap of n-grams and longest common subsequences between generated and reference texts. BLEU evaluates the alignment between generated and reference texts by calculating the precision of overlapping n-grams, with adjustments for brevity, effectively measuring linguistic accuracy and fluency.

CONTENT

FIGURES

TABLES

ACRONYMS AND ABBREVIATIONS

A100	NVIDIA A100 Tensor Core GPU
AI	Artificial Intelligence
Amazon AWS	Amazon Web Services
API	Application Programming Interface
BERTScore	Bidirectional Encoder Representations from Transformers Score
bf16	Brain Floating Point 16-bit (precision format used in AI models)
BLEU	Bilingual Evaluation Understudy
BLOB	Binary Large Object
BM25	Best Matching 25
ETL	Extract, Transform, Load
fp16	Floating Point 16-bit
GB	Gigabyte
Google GCP	Google Cloud Platform
GPU	Graphics Processing Unit
GPT	Generative Pre-trained Transformer
HTTP	Hypertext Transfer Protocol
LLM	Large Language Model
LORA	Low-Rank Adaptation
Microsoft Azure	Microsoft's cloud computing platform
MMLU Benchmark	Massive Multitask Language Understanding Benchmark (used for evaluating a model's performance across a wide range of subjects)
MTEB Benchmark	Massive Text Embedding Benchmark (used to evaluate text embedding models across a variety of tasks)
NF4	NormalFloat4 (quantization technique)
NLP	Natural Language Processing
OOM	Out of memory
PDF	Portable Document Format
PEFT	Parameter-Efficient Fine-Tuning
QLoRA	Quantized Low Rank Adaptation
RAG	Retrieval-Augmented Generation
RLHF	Reinforcement Learning from Human Feedback
ROUGE-1	Recall-Oriented Understudy for Gisting Evaluation (unigram overlap)
ROUGE-2	Recall-Oriented Understudy for Gisting Evaluation (bigram overlap)
ROUGE-L	Recall-Oriented Understudy for Gisting Evaluation (longest common subsequence)
SAT	Sistema de Administración Tributaria
SME	Subject Matter Expert
SQL	Structured Query Language
Word2Vec	A neural network-based model for generating word embeddings, which represent words as continuous vectors in a high-dimensional space. Words that are contextually or semantically similar are mapped to nearby vectors.

1. INTRODUCTION

***Abstract**: This chapter provides the background and justification for improving LLMs (Large Language Models) within specialized Spanish domains. It introduces the central problem and presents the hypothesis that combining QLoRA (Quantized Low-Rank Adaptation) with RAG (Retriever Augmented Generation) can significantly improve results in the context of specialized Spanish knowledge.*

1.1. Background

LLMs face challenges when applied to specialized domains. Pre-trained LLMs often fail to provide accurate and contextually relevant responses in these fields, necessitating techniques that improve performance without the high costs of training models from scratch.

Recent advances like QLoRA offer efficient fine-tuning, reducing resource demands while maintaining strong performance. Additionally, domain specialization has emerged as a key approach for adapting LLMs to complex tasks, underscoring the critical role of tailored techniques for domain-specific applications.

Moreover, RAG has been found to consistently outperform traditional fine-tuning methods, particularly in knowledge-intensive domains.

This work builds on these advancements by exploring the combination of QLoRA and RAG to enhance LLM performance in the context of specialized domain knowledge in the Spanish language.

1.2. Justification

This work is conducted in response to the growing recognition that LLMs struggle to manage the specialized complexities inherent in Spanish-language domains. These models frequently fail to provide accurate and contextually appropriate responses when addressing domain-specific inquiries. This performance gap highlights the necessity of developing enhanced methods for improving language models in specialized fields.

The growing complexity and specificity of professional domains have led to a heightened demand for more accurate, reliable, and accessible language tools in specialized sectors. To meet these challenges, this work is dedicated to enhancing LLM performance in specialized Spanish-language domains.

1.3. Problem

The recent rise of LLM has opened the possibility of using these tools in productive sectors where the massive handling of texts is part of daily life. One of these sectors is related to the Mexican tax legal framework, which demands a large amount of time and effort from professionals in this area to keep their knowledge up to date. Pre-trained LLMs perform poorly in specialized knowledge domains like the one mentioned above. This lack of precision hampers their ability to provide relevant and accurate information retrieval and legal interpretations.

Training a language model from scratch to improve its performance in this domain is highly resource-intensive and costly, making it impractical for many organizations. The challenge lies in determining effective methods to enhance the performance of LLMs in specialized knowledge domains.

1.4. Hypothesis

The hypothesis of this work is grounded in the premise that combining fine-tuning with the integration of relevant, domain-specific knowledge through RAG will enhance the model's predictive capabilities and contextual responsiveness. By leveraging QLoRA for efficient fine-tuning and RAG for incorporating specialized information, the model is expected to achieve significantly improved accuracy and contextual relevance in specialized domains.

To test this hypothesis, the following experiments will be executed using the same set of instructions and the same evaluation metrics:

1. Vanilla LLM applied to the specialized Spanish domain (no RAG, no fine-tuning).
2. Vanilla LLM with RAG applied to the specialized Spanish domain (no fine-tuning).
3. LLM fine-tuned with QLoRA applied to the specialized Spanish domain (no RAG).
4. LLM fine-tuned with QLoRA and augmented with RAG applied to the specialized Spanish domain.

1.5. Objectives

1.5.1. General Objective

To improve the performance of a pre-trained LLM when making inferences on unseen data instructions using zero-shot prompts in specialized knowledge domains in the Spanish language.

1.5.2. Specific Objectives

i) To select the data that will be used during the experiments and to design and implement the ETL process.
ii) To select the AI models to perform auxiliary tasks such as reading, extracting and summarizing data from PDF documents.
iii) To select the LLM which will be fine-tuned through PEFT QLoRA.
iv) To create the RAG module using a naive workflow.
v) To fine-tune the LLM using PEFT QLoRA.
vi) To conduct experiments and compute evaluation metrics.
vii) To compare the performance of each experiment and draw conclusions.

1.6. Scientific, technological novelty or contribution

This work's contribution lies in designing a scalable, reliable, and maintainable architecture to execute the ETL process specific to the input formats handled by Mexican tax laws; future works or adaptations can build improvements or can extend the current architecture. This work also provides evidence to the existing discussion of what combination of techniques can improve the performance of an LLM in specialized domains, in this case, adding the constraint of the Spanish language.

2. STATE OF THE ART

Abstract: *This chapter provides a brief overview of the state of the art of the topics on which this work was developed.*

2.1. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks

In the paper titled "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" [1], the authors explore the integration of RAG (Retrieval-Augmented Generation) with pre-trained LLMs (Large Language Models) to enhance their performance on knowledge-intensive NLP (Natural Language Processing) tasks. Traditional large models store extensive information but struggle with precision in accessing and manipulating this knowledge, often lagging specialized architectures in task-specific applications. To address these limitations, the study introduces RAG models that employ parametric (pre-trained seq2seq models) and non-parametric (dense vector index of Wikipedia) memories.

The study evaluates RAG models across a variety of tasks, particularly highlighting their superiority in open-domain question answering. There, they set new performance benchmarks, outperforming both purely parametric models and combined retrieve-and-extract architectures [1]. The RAG models demonstrated the ability to generate more specific, diverse, and factually accurate language compared to a state-of-the-art parametric-only baseline.

Further analysis revealed that RAG models could generate correct answers even without verbatim evidence in the retrieved documents, showcasing their utility over extractive methods which require exact matches in source documents. RAG models still managed notable accuracy in scenarios where the correct answers were not present in any retrieved document, thereby underscoring their robustness and the effectiveness of blending generative and retrieval capabilities [1].

Additionally, the research investigates the impact of different retrieval mechanisms, showing that learned retrieval significantly enhances performance across tasks compared to basic keyword overlap methods like BM25 (Best Matching 25) [1]. The flexibility of RAG to incorporate updated knowledge dynamically at test time without retraining also highlights its potential for practical applications where information evolves or expands frequently.

Overall, this study establishes RAG as a potent tool for improving the performance of language models on knowledge-driven tasks, advocating for its use in applications where accuracy and depth of knowledge are paramount [1].

2.2. Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning

In the paper titled "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning" [2], published in March 2023, the authors review parameter-efficient fine-tuning (PEFT) methods for large language models, examining over 40 studies published from February 2019 to February 2023. It addresses the challenges of traditional fine-tuning, which requires adjusting numerous parameters, thus becoming impractical for large-scale models. The authors introduce a taxonomy to categorize PEFT approaches based on whether they introduce new parameters or fine-tune existing subsets. This classification encompasses over 30 distinct PEFT methods.

The paper elaborates on various PEFT strategies, prominently featuring additive and selective methods [2]. Additive methods include the introduction of small networks or layers to the existing model framework, exemplified by adapter modules and soft prompts. These techniques focus on optimizing the model without extensively altering all parameters, thus preserving computational resources and improving training efficiency.

Selective methods fine-tune specific parts of a model, such as the top layers or specific biases, often based on their relevance to the task. This targeted approach reduces the memory footprint and enhances the adaptability of the model to new tasks with minimal retraining [2].

Reparametrization-based methods are also discussed, utilizing low-rank representations to decrease the number of trainable parameters significantly. Techniques like LoRA (Low-Rank Adaptation) enable updates through simplified matrix decompositions, allowing for scalability in models with billions of parameters [2].

Hybrid methods combine multiple PEFT strategies to leverage their collective strengths, achieving high accuracy while maintaining parameter efficiency. The paper concludes with best practices for future research, emphasizing the need for explicit reporting of parameter counts, evaluations across various model sizes, and direct comparisons between methods. It advocates for standardized benchmarks and competitions to foster a consistent evaluation framework, enhancing the understanding and development of PEFT methods [2].

2.3. QLoRA: Efficient Finetuning of Quantized LLMs

In the paper titled "QLoRA: Efficient Finetuning of Quantized LLMs" [3], published in May 2023 by the University of Washington, the authors present a study mentioning that QLoRA (Quantized Low-Rank Adaptation) is a technique that represents a significant advancement in the field of LLM fine-tuning. According to the research, this method "reduces memory usage enough to fine-tune a 65B parameter model on a single 48GB GPU while preserving full 16-bit fine-tuning task performance" [3].

QLoRA utilizes LoRA combined with 4-bit quantization to achieve these results. By implementing innovations such as 4-bit NormalFloat (NF4) and double quantization, QLoRA enables the fine-tuning of models with billions of parameters using relatively modest hardware resources [3].

This approach addresses one of the core challenges in fine-tuning LLMs: resource constraints. The study shows that QLoRA can achieve performance levels close to ChatGPT with only 24 hours of fine-tuning on a single GPU [3]. The researchers also note that the technique significantly broadens access to state-of-the-art NLP technology, making it more accessible to smaller research teams and independent researchers who may lack the resources of larger organizations [3].

2.4. Domain Specialization as the Key to Make Large Language Models Disruptive: A Comprehensive Survey

In the paper titled "Domain Specialization as the Key to Make Large Language Models Disruptive: A Comprehensive Survey" [4], published in October 2023, the authors provide a survey of techniques used to specialize LLMs for particular domains. The study categorizes the various methods based on their accessibility to the underlying LLMs, offering a detailed taxonomy of domain specialization techniques [4].

This survey highlights the growing need for domain-specific LLMs, as applying general models to specialized fields presents significant challenges due to the complexity and heterogeneity of domain data. The researchers identify key application areas that could benefit from domain-specialized LLMs, emphasizing that techniques such as knowledge elicitation and model adaptation can play a pivotal role in improving the applicability of LLMs across domains [4].

The paper concludes by identifying open challenges, such as the lack of domain-specific expertise and the complexity involved in adapting models to meet the stringent requirements of specific fields [4].

2.5. Knowledge Injection: Fine-Tuning vs. Retrieval-Augmented Generation (RAG)

In the paper titled "Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs" [5], published in January 2024, the authors compare two standard methods for incorporating external information into LLMs: unsupervised fine-tuning and RAG. The study finds that while fine-tuning can improve model performance to some extent, RAG consistently outperforms it, particularly in tasks that require access to new or specialized knowledge [5].

RAG enables LLMs to dynamically retrieve and integrate relevant information during the generation process, allowing models to handle knowledge that may not be explicitly encoded during training [5]. The study suggests that RAG is a more effective method for knowledge injection, particularly in domains where the information is constantly evolving. However, the authors also point out that further research is needed to explore combinations of fine-tuning and retrieval-based methods to enhance LLMs' ability to process complex domain-specific knowledge [5].

3. THEORETICAL FRAMEWORK

Abstract: *This chapter presents the theoretical framework underlying the study. It covers LLMs, Prompt Engineering, the ETL Medallion Architecture, Embedding Models, Semantic Search, RAG and PEFT QLoRA along with the evaluation metrics used to assess the LLM performance.*

3.1. Mexican Tax Law

Tax laws are legal regulations that establish guidelines for the collection of taxes and other fiscal revenues. They define the subject, base, and rate for the proper fulfillment of tax obligations. Legal provisions are governed by Laws, Regulations, Codes, Decrees, Miscellaneous Resolutions, and Normative Criteria, among others, which can be found on the portal of Sistema de Administración Tributaria (SAT) [6].

3.2. Large Language Models

LLMs (Large Language Models) are advanced AI models designed to process and generate human language. They rely on deep neural networks, particularly the Transformer architecture, which allows models to handle extensive amounts of textual data. The Transformer architecture, introduced in 2017 in the paper "Attention is All You Need" [8], revolutionized the field by replacing RNNs (Recurrent Neural Networks) with attention mechanisms, enabling faster training and more accurate performance for text generation tasks.

3.2.1. Evolution of LLMs

The term Language AI refers to technologies that process natural language, often overlapping with NLP (Natural Language Processing) [9]. LLMs have evolved from earlier models like the bag-of-words method, which represented text as word counts, to more advanced models like Word2Vec, introduced in 2013 [9]. Word2Vec captured semantic meaning by training on large corpora, generating vector representations (embeddings) that indicate relationships between words based on their context. However, Word2Vec struggled with longer sequences of text and could not handle ambiguity effectively [9].

The introduction of attention mechanisms significantly improved how language models manage sequences, allowing the model to focus on relevant parts of the input when generating or understanding text [8]. This evolution culminated in the Transformer architecture, which employs self-attention layers, enabling models to process entire sequences of text simultaneously rather than step by step as with RNNs [8].

3.2.2. Public vs. Private LLMs

One important distinction in the field of LLMs is between public and private models. Private LLMs, such as OpenAI's GPT-4 or Anthropic’s Claude, are proprietary models developed by organizations that keep their internal architecture and training data confidential [9]. These models are typically accessed via APIs, allowing users to benefit from high performance without the need for substantial hardware resources. However, this also limits user control and customization, as fine-tuning or modifying the models is often not allowed. Additionally, concerns about privacy and the use of sensitive data in these systems can arise, particularly in regulated industries like healthcare [9].

In contrast, public LLMs like Meta’s LLaMA and Hugging Face’s open models provide access to their weights and architecture, allowing users to run these models on local hardware and even fine-tune them for specific tasks [9]. While this grants more flexibility, public models often require substantial computational resources, and users must manage the hardware and setup themselves [9]. Public models

also foster collaboration through large open-source communities, enabling faster innovation and broader accessibility to cutting-edge AI technology [7].

3.2.3. LLMs applications

LLMs have diverse applications. These models can classify text, generate dialogue, retrieve relevant documents, and more [9]. For example, RAG integrates external knowledge bases with LLMs, enhancing their ability to provide accurate responses by retrieving and leveraging external information [7]. Additionally, prompt engineering techniques allow users to customize the inputs given to LLMs to optimize their outputs for various tasks [7].

In summary, LLMs are central to advancements in NLP, with the Transformer architecture powering their capabilities [8]. Public and private LLMs each have unique advantages, offering flexibility, control, or performance depending on user needs [9].

3.3. Prompt Engineering

Prompt engineering seeks to optimize the interaction between humans and LLMs through the careful design of prompts. By crafting effective prompts, users can guide LLMs towards desired outcomes and maximize their potential. The Prompt Engineering Guide provides a structured framework for understanding key concepts, techniques, and applications in this field [11].

Key Concepts:

- Prompt: A textual input that serves as a query or instruction for an LLM.
- Zero-shot prompting: Directly instructing an LLM without providing examples.

```
Classify the text into neutral, negative, or positive.

Text: Commander Shepard's speech inspired the crew before the final mission.
Sentiment:
```

- Few-shot prompting: Providing a few examples to guide the LLM.

```
To perform a "gambitwist" in chess means to sacrifice a piece with the hope of gaining a
tactical advantage later. An example of a sentence that uses the word gambitwist is: During
the tournament, Alex executed a daring gambitwist by sacrificing their bishop, which later
led to a decisive checkmate.

A "rookslide" in chess is a maneuver where a rook moves swiftly along an open file to
dominate the board. An example of a sentence that uses the word rookslide is:
```

These techniques are fundamental to improving LLM performance and adaptability in various tasks [11].

Prompt Engineering techniques

- Prompt design: Crafting clear, concise, and informative prompts.
- Prompt optimization: Experimenting with different prompt formats and structures.
- Contextualization: Incorporating relevant background information into prompts.
- Task specification: Clearly defining the desired output or outcome.

Each of these strategies plays a role in maximizing the effectiveness of prompts across various AI applications [11].

Prompt Engineering applications:

- Question answering: Generating informative and accurate responses to user queries.
- Text summarization: Condensing long texts into concise summaries.
- Creative writing: Assisting in the generation of creative content, such as stories or poems.
- Translation: Translating text from one language to another.
- Code generation: Generating code snippets based on natural language descriptions.

These applications highlight the versatility of prompt engineering in enhancing LLM capabilities across multiple domains [11].

3.4. Medallion architecture

Medallion Architecture is a structured approach to data organization that uses multiple layers to enhance data quality as it progresses through various stages of processing [12].

The bronze layer stores raw data in its original state, capturing all historical data and metadata. This ensures that the organization has access to complete, unaltered records.

The silver layer refines the data by validating and deduplicating it, making it more reliable for analysis. This layer is essential for ensuring that downstream processes have access to clean, trustworthy data.

The gold layer involves further transformation and aggregation of data, which is now optimized for analytics and ML (Machine Learning) applications. At this stage, the data is refined into what are often called golden documents, representing high-quality, actionable information. These documents are crucial for powering advanced analytical processes, including training machine learning models and driving decision-making systems [12].

By progressively improving the data through each layer, the Medallion Architecture ensures that the final output is suitable for complex analytical tasks, including ML models, which require clean, well-structured data to produce accurate results [12].

3.5. Tokens and Embeddings

Tokens and embeddings are foundational concepts for understanding and working with LLMs. Tokens are small chunks of text, such as words or parts of words, that models break down for processing. A key part of how LLMs function is through tokenization, the process by which input text is split into tokens before the model processes it. For example, in GPT-4, text input is broken into tokens that can represent complete words or partial words, which are then translated into numeric representations (embeddings) that the model uses for computation [9].

Embeddings are vectors that represent tokens numerically, capturing their meaning in a way that the model can interpret. These embeddings allow the model to measure semantic similarity between tokens, which is essential for understanding context and meaning. For example, the model can determine that the words "apology" and "apologize" are related through similar embeddings. Modern models, like Word2Vec, introduced in 2013, were some of the first to generate meaningful embeddings by training on large amounts of text, learning to place semantically similar words closer together in vector space [9].

LLMs generate contextualized embeddings, meaning that the representation of a word changes based on the context in which it appears. This allows the model to understand the nuanced meaning of words that have multiple interpretations, such as "bank" (which could mean a financial institution or the side of a river) [9].

Text embeddings, which represent entire sentences or documents as single vectors, are widely used in applications like semantic search and recommendation systems. These embeddings capture the overall meaning of longer texts, enabling LLMs to perform tasks like topic modeling and text classification [9].

In practical terms, embedding models are crucial for applications beyond text processing, such as in recommendation systems and even in image generation models like DALL·E. By assigning meaningful numerical representations to words, tokens, or even multimedia, embeddings power a wide range of machine learning applications [9].

3.6. Retriever-Augmented Generation (RAG)

RAG is a method that enhances LLMs by incorporating external knowledge sources, like databases. This helps address issues such as factual inaccuracies and hallucinations. RAG is particularly useful for knowledge-intensive tasks and domains with constantly evolving information. Unlike traditional approaches, RAG doesn't require retraining the LLM for specific applications [16].

Key components:

- **Input**: The question or prompt.
- **Indexing**: Organizing external knowledge into a searchable format.
- **Retrieval**: Finding relevant information from the indexed knowledge.
- **Generation**: Using the retrieved information to generate a response.

Paradigms:

- **Naive RAG**: A straightforward approach, but prone to issues like low precision and recall.
- **Advanced RAG**: Improves retrieval quality through optimization techniques.
- **Modular RAG**: Offers flexibility and customization by allowing for different modules.

Framework:

- Retrieval: Retrieval techniques and optimization.
- Generation: Techniques for generating coherent text.
- Augmentation: Integrating retrieved context into the generation process.

RAG is a valuable tool for improving the accuracy and reliability of LLMs, especially in knowledge-intensive domains. By effectively combining retrieval and generation, RAG can help LLMs provide more informative and relevant responses [16].

3.7. PEFT QLoRA

3.7.1. Parameter-Efficient Fine-Tuning (PEFT)

PEFT (Parameter-Efficient Fine-Tuning) is a collection of techniques designed to efficiently adapt large pre-trained language models (LLMs) to specific downstream tasks. Unlike traditional fine-tuning, which updates all model parameters, PEFT methods focus on training a smaller subset of parameters, significantly reducing computational costs and memory requirements [2]. This allows for the fine-tuning of even the largest language models on standard hardware.

Benefits:

- Reduced training time: By training fewer parameters, the training process becomes faster.
- Lower memory footprint: Smaller models can be trained on devices with limited memory.
- Ability to fine-tune larger models: Even the most massive language models can be adapted to specific tasks [2].

Techniques:

- Additive methods: These methods introduce new parameters to the model, such as adapters or soft prompts, which are then trained.
- Selective methods: Only a subset of existing model parameters is fine-tuned, often based on layer type or importance.
- Reparameterization-based methods: These methods use low-rank approximations to represent the weight updates, reducing the number of trainable parameters [2].

3.7.2. QLoRA

QLORA (Quantized Low-Rank Adaptation) is a reparametrization-based PEFT that works by introducing low-rank matrices to the original weight matrices, enabling efficient fine-tuning while preserving the overall model architecture using quantization [3].

Quantization is the process of reducing the numerical precision of data. In the context of neural networks, this involves representing the weights of the network with fewer bits. For instance, instead of using 32-bit floating-point numbers (float32), we can use 8-bit integers or even fewer [19].

Quantization allows:

- Model size reduction: Smaller models require less memory and can be deployed on devices with limited resources.
- Energy efficiency: Operations with lower precision numbers are computationally cheaper.
- Hardware acceleration: Many hardware accelerators (like GPUs and TPUs) are optimized for quantized operations [19].

3.7.2.1. QLoRA and 4-bit Quantization

QLoRA employs a specific quantization technique called 4-bit NormalFloat (NF4). This method is particularly well-suited for neural network weights, which often follows a normal distribution [3]. NF4 assigns the most frequent values to a limited number of quantization levels, minimizing information loss.

How does QLoRA work with quantization?

1. Quantization: The pre-trained language model is quantized to 4 bits, significantly reducing its size.
2. Low-Rank Adaptation (LoRA): Small modules called adapters are added to the quantized neural network. These adapters are the only parameters that are fine-tuned during training.
3. Training: The adapters are trained to compensate for the loss of precision caused by quantization [3][19].

This technique enables training language models with billions of parameters on a single GPU achieving results comparable to models trained with full precision [3].

3.8. Evaluation metrics for LLM

Word-level metrics are commonly used in Natural Language Processing (NLP) to evaluate the quality of tasks such as text generation, machine translation, and summarization. These metrics compare a text generated by a model to a reference text to determine their similarity. Among the most popular metrics are BERTScore, ROUGE, and BLEU.

BERTScore leverages the ability of bidirectional language models like BERT to understand the meaning of words and phrases, calculating a semantic similarity score between the generated and reference texts. Unlike n-gram-based metrics, BERTScore is more robust to rephrasing and paraphrasing, making it more suitable for evaluating the semantics of text [22]. ROUGE (Recall-Oriented Understudy for Gisting Evaluation), on the other hand, is based on the matching of n-grams, that is, sequences of n consecutive words. ROUGE-1 considers unigrams, ROUGE-2 bigrams, and ROUGE-L the longest common subsequence [23]. BLEU (Bilingual Evaluation Understudy) is like ROUGE but also considers the precision of n-grams and penalizes the repetition of phrases [24]. Although both ROUGE and BLEU are widely used metrics, they are more sensitive to syntax and word choice than BERTScore.

4. METHODOLOGICAL DEVELOPMENT

Abstract: *This chapter presents the methodology used to run experiments. It covers model selection, the creation of a synthetic dataset based on Mexican Tax Law, the implementation of the RAG module, the application of PEFT QLoRA for fine-tuning, and the methods to evaluate the LLM performance.*

4.1. Define use cases

The primary goal of this project is to enhance the performance of a LLM in generating accurate and contextually relevant responses in the domain of Mexican tax law. This involves the next use cases:

4.1.1. Preparing a Synthetic Dataset

4.1.1.1. Actors

- AI engineer

4.1.1.2. Preconditions

Access to raw text data (PDF documents of Mexican tax laws).

4.1.1.3. Main success scenario

- The actor uploads raw PDF documents to the system.
- The system extracts text from the PDFs using Azure AI Documents.
- The system uses a generative model (e.g., Chat-GPT-3.5 Turbo) to create synthetic instructions and responses based on the extracted text.
- The synthetic dataset is stored in a structured format (JSON) in CosmosDB.
- The actor reviews and verifies the quality of the synthetic dataset.

4.1.1.4. Assumptions

- The generative model can create relevant and accurate synthetic data.
- The system has sufficient resources to process the PDFs and generate the dataset.

4.1.1.5. Open issues

- Ensuring the synthetic data accurately represents the complexities of Mexican tax law.
- Continuous improvement of the synthetic data generation process.

4.1.2. Create RAG module

4.1.2.1. Actors

- AI engineer

4.1.2.2. Preconditions

- Access to the text-embeddings-ada-002 model through the OpenAI API
- Availability of Pinecone as the vector database.

4.1.2.3. Main success scenario

- The engineer sets up the environment to use the OpenAI embeddings API
- The engineer configures the Pinecone database to store embedding vectors with the needed dimensions based on text-embedding-ada-002 specification
- The system processes the instructions dataset, generating embeddings for each article
- The generated embeddings are stored in Pinecone with appropriate indexing.

- The engineer implements a semantic search function that uses cosine similarity to retrieve relevant embeddings from Pinecone based on a given query.
- The retrieved embeddings are used to augment the LLM's generation process, providing contextually relevant information to enhance the response.

4.1.2.4. Assumptions
- The text-embeddings-ada-002 model can generate high-quality embeddings for the given dataset.
- Pinecone is properly configured and accessible for storing and retrieving embedding vectors.

4.1.2.5. Open issues
- Optimizing the retrieval process for efficiency and accuracy.

4.1.3. Model fine-tuning

4.1.3.1. Actors
- AI engineer

4.1.3.2. Preconditions
- Availability of synthetic datasets.
- Access to fine-tuning tools and computational resources (Google Colab GPU A100)

4.1.3.3. Main success scenario
- The AI engineer loads the synthetic dataset into the fine-tuning tool.
- The engineer configures the fine-tuning parameters (e.g., learning rate, batch size).
- The fine-tuning process is executed on TPUs, leveraging Google Colab's infrastructure.
- The fine-tuned model is evaluated on a validation dataset.
- If needed, the AI engineer adjusts the parameters and retry the process.

4.1.3.4. Assumptions
- The fine-tuning tools and computational resources are available and properly configured.
- The synthetic dataset is of high quality and representative of the domain.

4.1.3.5. Open issues
- Optimizing fine-tuning parameters for the best performance.
- Managing computational resources efficiently.

4.1.4. Evaluating Model Performance

4.1.4.1. Actors

- AI engineer

4.1.4.2. Preconditions

- Access to evaluation tools and the ground truth dataset.
- Availability of vanilla, RAG, and fine-tuned models.

4.1.4.3. Main success scenario

- The system evaluates the vanilla model using BERTScore, ROUGE (1,2, L) and BLEU.
- The system evaluates the vanilla model + RAG using the same metrics.
- The system evaluates the fine-tuned model using the same metrics.
- The system evaluates the fine-tuned model + RAG using the same metrics.
- The results are compared and analyzed to draw conclusions.

4.1.4.4. Assumptions

- The evaluation tools and datasets are reliable and accurately reflect the model's performance.

4.1.4.5. Open issues

- Ensuring the evaluation dataset is comprehensive and representative of real-world use cases.
- Interpreting the evaluation metrics accurately to inform further model improvements.

4.2. Choosing the models

4.2.1. Text embeddings model

For the text embeddings model, the OpenAI text-embedding-ada-002 was selected. This model was chosen due to its quick implementation via an HTTP API, which significantly reduces the setup time and accelerates processes that require embedding tasks. Additionally, the model is cost-effective, reliable, and has demonstrated good performance on the MTEB benchmark.

4.2.2. Extraction, summarization, and text generation model

To handle the tasks of extracting content from unstructured corpus and generating synthetic instructions and responses based on the text of tax law articles, we employed both the OpenAI GPT-3.5 Turbo model and the Azure AI Document Intelligence Layout Model. The process begins with the Document Intelligence Layout Model, which performs an initial extraction of text lines and bounding boxes from the unstructured documents. Using this extracted data, the GPT-3.5 Turbo model is then employed through prompt engineering to extract and structure each individual article from the tax law under analysis which finally this article data is used to generate a synthetic dataset of instructions.

Both models were selected due to their quick implementation, significantly reducing setup time and accelerating the implementation processes.

4.2.3. Inference model (main model)

To select the large language model candidates the next constraints were taken into consideration:

- The model must allow fine-tuning tasks.
- The model must be large size (1 billion – 10 billion parameters) showing good performance on unseen data and zero-shot prompts.
- The model must be available in Hugging Face. This platform was chosen for the next reasons:
    - Ease of use,
    - Fine-tuning capabilities,
    - Extensive community support and contributions,
    - Integration with PyTorch.

The chosen model for this project is **google/gemma-7b-it**. Several factors influenced the selection of this model:

- **Community Support**: google/gemma-7b-it benefits from extensive community support, providing extensive resources and documentation.
- **Ease of Use for Inferences**: The model is user-friendly, facilitating straightforward implementation for inference tasks.
- **Versatility for Fine-Tuning**: Its versatility allows for effective fine-tuning, whether in Google Colab for experimentation or Amazon AWS/ Microsoft Azure /Google GCP for enterprise and production purposes.
- **Implementation Flexibility**: The model supports a wide range of potential implementations using different libraries, such as PyTorch, enhancing its adaptability to various needs and environments.
- **Language Proficiency**: It has demonstrated proficiency in processing multilingual texts, making it suitable for the Spanish language (based on MMLU benchmark results)
- **Model Size and Capabilities**: As a 7-billion-parameter model, enabling robust natural language understanding and generation.

4.3. Data preparation

Figure 1 provides a high-level overview of the ETL process used to transform raw PDF documents into structured golden documents. These processed documents serve two key purposes: first, they are used to generate a semantic search index in Pinecone by creating embeddings that enable efficient retrieval; second, they act as input for the synthetic generation of an instruction dataset, which is essential for fine-

tuning the model. This workflow ensures that the processed data is both searchable and suitable for model adaptation, optimizing performance in downstream tasks.

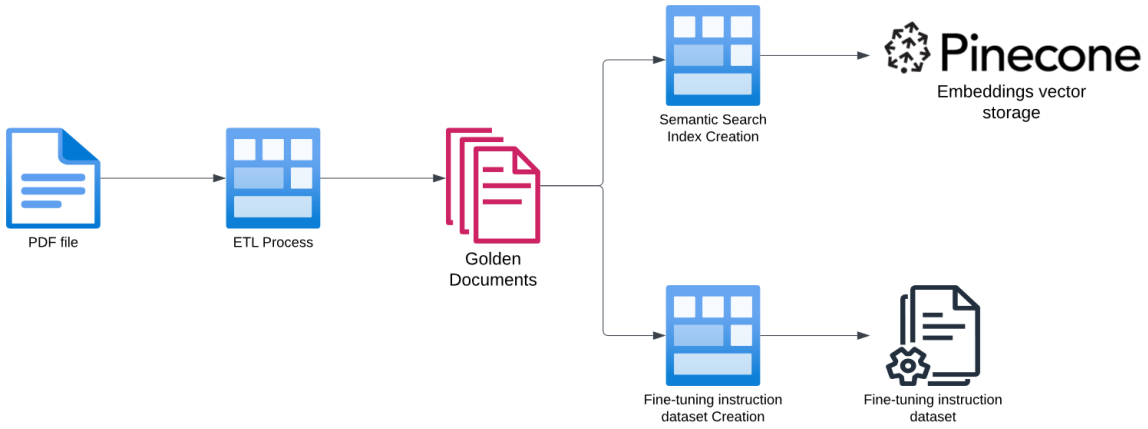

Figure 1. Data preparation process

4.3.1. Selecting the data

Only "La Ley de Impuesto sobre la Renta 2024 (LISR 2024)" was used for experimentation and validation of this work's hypothesis. In future iteration, all laws must be processed to obtain a useful model for real world scenarios. Figure 2 presents the Mexican tax legal framework.

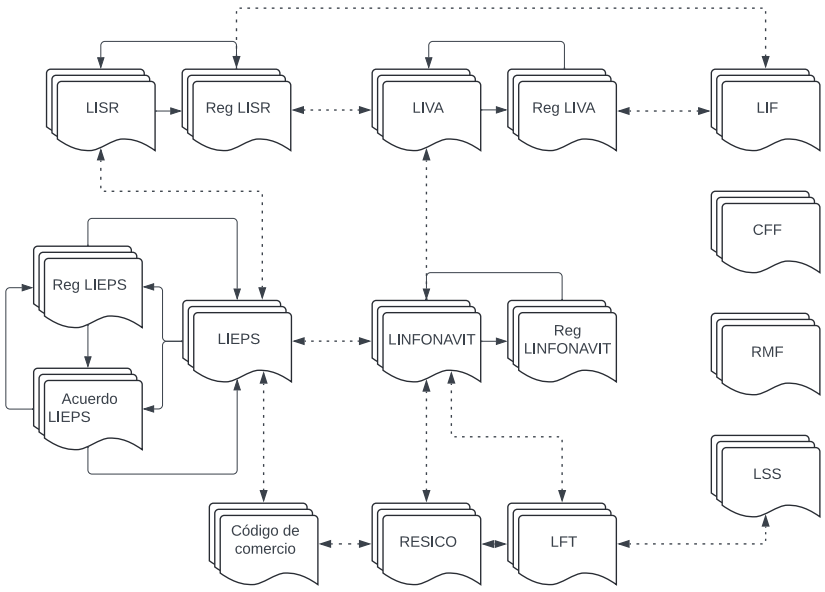

Figure 2. Mexican tax law corpora

4.3.2. ETL process development

The ETL process for this project is designed to follow the medallion architecture, consisting of three stages (bronze, silver and golden). The process is performed asynchronously, handling each PDF file individually. The initial state of the data contains headers, separator lines, text, headers, footers, and page numbers.

Once the ETL process is triggered, bronze documents are generated. Each of these documents represents a page from the original file, and each document contains information about the lines on a page, with each line associated with a set of coordinates (x, y) that define a bounding box for its location in the original file. Subsequently, silver documents are generated, where each document represents a law article. Finally, gold documents are created, which contain the data from both the bronze and silver documents. Additionally, two sets of instructions are included: one for creating the semantic search index, which will be a component of the RAG subsystem, and the other for creating a training dataset.

Figure 3 presents the data transformation stages, starting with the bronze state, and ending with the golden state.

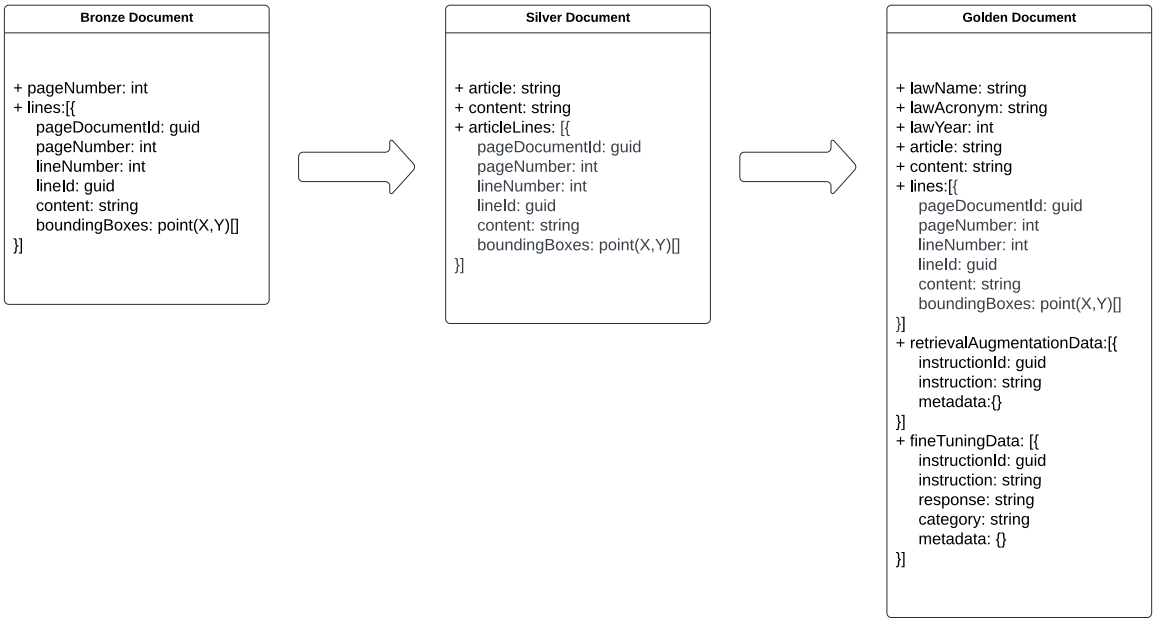

Figure 3. Medallion document stages

The ETL process was implemented using a cloud-native serverless architecture deployed in Microsoft Azure. Figure 4 presents the deployment architecture. The detailed execution flow is presented below.

0a. Upload the file to the backend service

0b. Save the PDF file in the BLOB storage

1a. The PDF file is read and analyzed using the Azure AI Documents service with the "layout document" option. This service extracts the text line by line, generating bounding boxes for each line.

1b. The extracted text and bounding box information are stored in the first stage (bronze document). Each JSON document represents a page of the PDF document.

2a. Using the OpenAI chat-GPT-3.5-Turbo LLM the individual articles are extracted from the bronze documents.

2b. The extracted articles are stored in the second stage (silver document). Each JSON document represents an article of the law, and each document contains the article text and all the bounding boxes for the article.

3a. Using the OpenAI chat-GPT-3.5-Turbo LLM a synthetic dataset is created with instructions categorized into open-questions, closed-questions, and summarizations using a supervised approach.

3b. The generated instructions are stored in the third stage (golden document). Each document contains two properties, one for RAG with only the instructions with no responses, and one second stores the instructions with responses, this last property will be the input to generate the fine-tuning dataset.

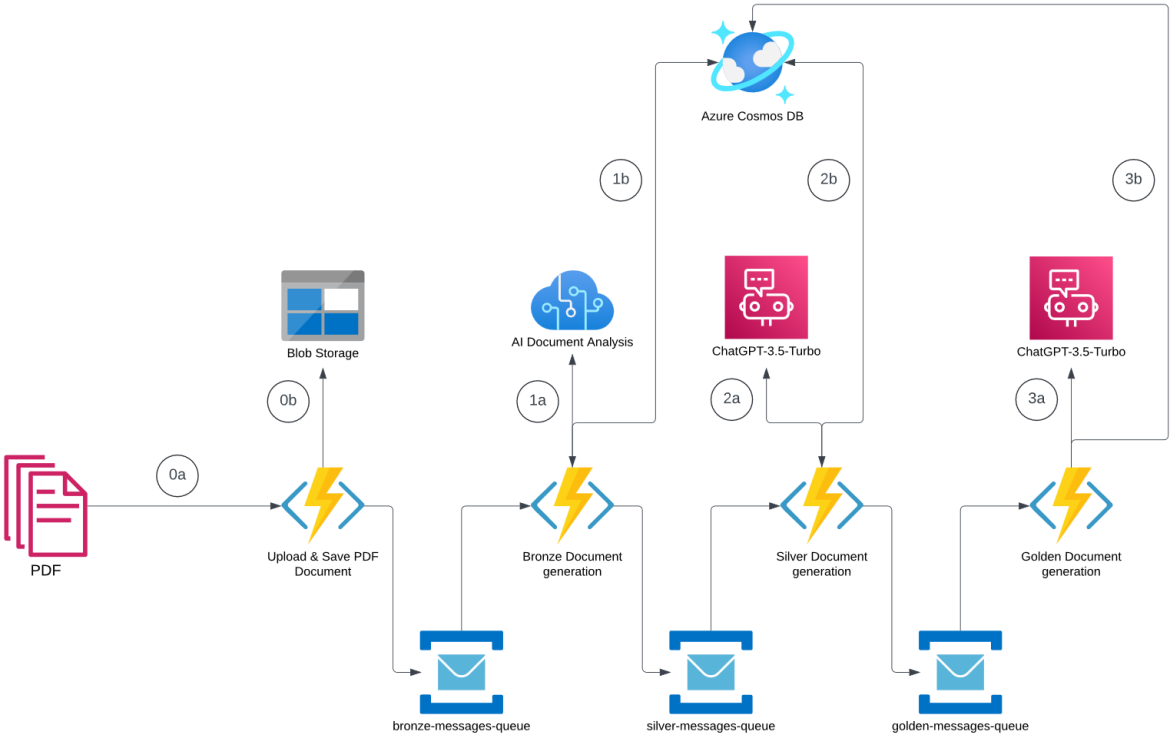

Figure 4. ETL process

4.3.2.1 Validation and Recommendations

For production implementations, it is recommended to involve a group of human subject matter experts to validate the silver and golden documents. This validation step would enhance the reliability of the data extraction process, ensuring the accuracy of the extracted data. Figure 5 presents an activity diagram that illustrates which steps are performed synthetically by AI and which steps could be added to incorporate a validation process by human subject matter experts to ensure the quality of the extracted text.

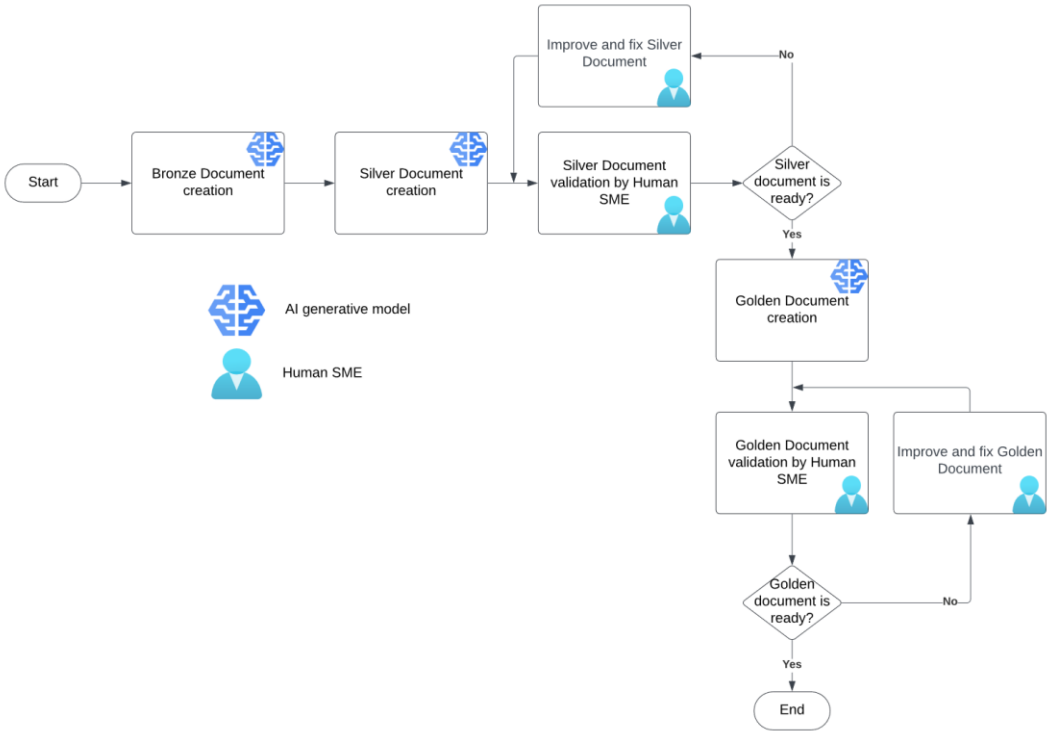

Figure 5. ETL process enhancements

4.3.3. Instruction dataset split

As a result of the ETL process, 183 documents were obtained, each corresponding to an article of the law "Ley de Impuesto sobre la Renta 2024". Each document includes summarization instruction and a variable number of instructions in the categories of closed-questions and open-questions. Figure 6 presents a Venn diagram illustrating how the data from the golden documents is divided. It is important to note that the testing data is not used in the RAG process, nor is it utilized for fine-tuning.

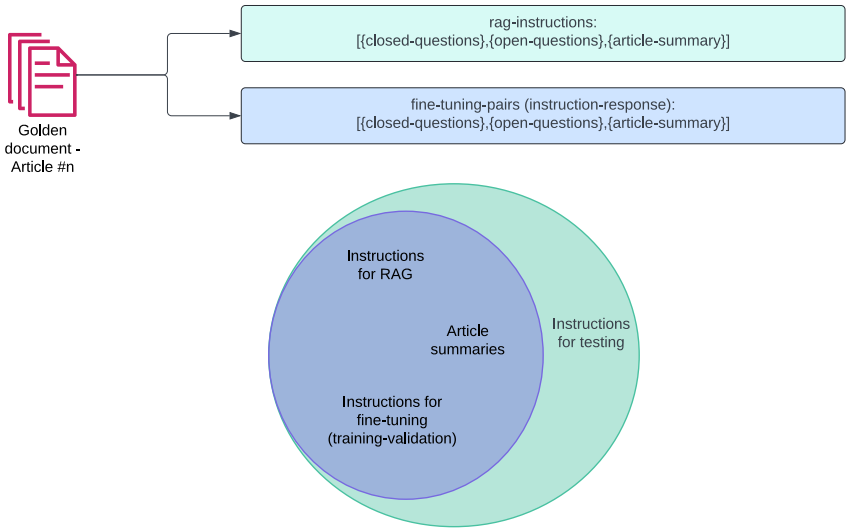

Figure 6 Instruction data partition Venn diagram

Figure 7 shows the split of instruction-response pairs for fine-tuning. Closed and open-question pairs follow a 70-30 ratio for training/validation and testing, while summarization instructions are fully used in both processes.

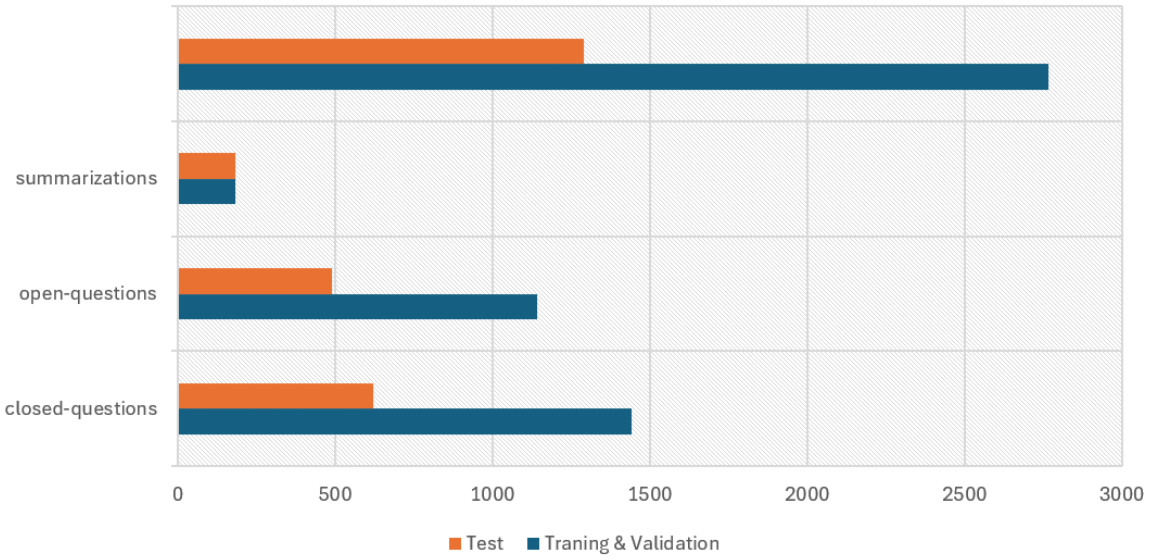

Figure 7. Dataset split comparison bar chart

4.3.4. Semantic search index creation

The vector index for semantic search was constructed using a set of instructions synthetically generated by the chat-gpt-3.5 turbo generative model. The hypothesis behind this approach is that the retrieval of

relevant information in response to user queries improves significantly if the index contains a wide variety of examples reflecting the types of queries users typically ask regarding the topic of interest. For this reason, the index was created using instructions that include closed-questions, open-questions, and summarizations.

As part of the metadata, identifiers are referenced to the golden documents, from which auxiliary data—such as article identifiers and legal references—can be retrieved to perform a complete information query. Furthermore, this metadata allows access to supplementary data, such as the bounding boxes of relevant information, enhancing the user experience in applications.

The index creation process is fully deployed in the cloud and follows these steps:

1. Retrieval of the instructions designated for the RAG module, which are stored in the golden documents.
2. Creation of the embedding vectors using the text-embedding-ada-002 model.
3. Generation of metadata for the embedding vectors.
4. Insertion of the vectors into the vector index.

Figure 8 illustrates the cloud deployment architecture in Azure for the module responsible for creating the vector index used in the retrieval component of the RAG implementation. The diagram depicts the use of a NoSQL database to store and retrieve golden documents, which serve as the foundation for indexing. An orchestrator process, implemented using Azure Functions, manages the workflow by invoking the OpenAI Embeddings API to generate vector representations and storing them in Pinecone DB

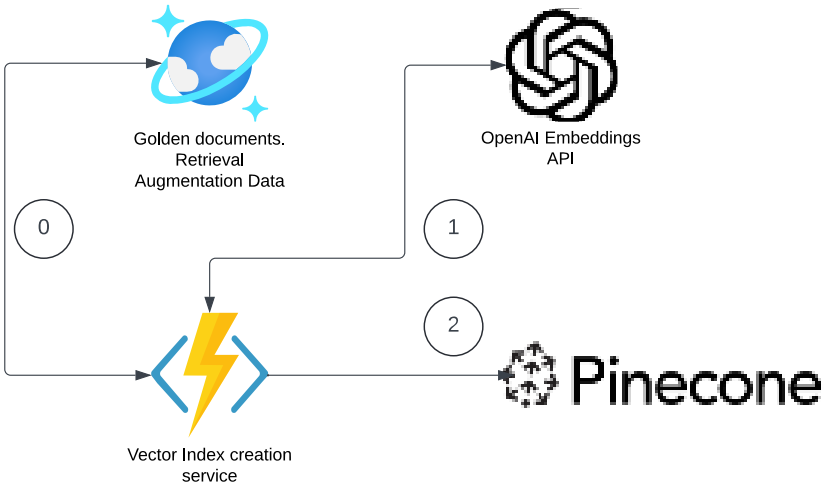

Figure 8. Semantic search index creation

4.3.5. Fine-tuning instruction dataset creation

The fine-tuning dataset was constructed using the subset of instructions designated for training-validation. From these instructions and responses, a total of 2,695 conversations were generated using tokens to define the roles of the user and the system. The format used for generating the conversations follows the structure corresponding to google/gemma-7b-it.

```
Instruction: ¿Qué personas están obligadas al pago del impuesto sobre la renta según el Artículo 1?

Response: Las personas físicas y las morales están obligadas al pago del impuesto sobre la renta en varios casos, como los residentes en México respecto de todos sus ingresos, los residentes en el extranjero con un establecimiento permanente en el país, y los residentes en el extranjero con ingresos de fuentes en territorio nacional.
```

Conversation example:

```
<bos><start_of_turn>user ¿Qué personas están obligadas al pago del impuesto sobre la renta según el Artículo 1?<end_of_turn><start_of_turn>model Las personas físicas y las morales están obligadas al pago del impuesto sobre la renta en varios casos, como los residentes en México respecto de todos sus ingresos, los residentes en el extranjero con un establecimiento permanente en el país, y los residentes en el extranjero con ingresos de fuentes en territorio nacional.<eos>
```

4.4. Experimentation

The purpose of these experiments is to validate the hypothesis by comparing the inference performance of different model configurations on a shared test set. The hypothesis being tested is that the performance of a language model in retrieving relevant and contextually accurate information improves significantly when using a combination of fine-tuning techniques (such as PEFT QLoRA) and RAG. The model's effectiveness will be measured through a set of predefined evaluation metrics.

The test dataset consists of a set of instructions designed to cover a variety of query types, such as closed-questions, open-questions, and summarizations. The dataset is distinct from the fine-tuning dataset, except for shared article summarizations.

Four different model configurations were tested under identical conditions, using the same test dataset:

- **Experiment 1**: Inference using LLM vanilla (no fine-tuning, no RAG).
- **Experiment 2**: Inference using LLM vanilla + RAG.
- **Experiment 3**: Inference using LLM fine-tuned with PEFT QLoRA (no RAG).
- **Experiment 4**: Inference using LLM fine-tuned with PEFT QLoRA + RAG.

Each experiment followed the same procedure for instruction processing:

1. Fetch the test instructions.
2. Process the instructions in batches of 50.
3. Generate responses using the model configuration.
4. Persist the responses for further evaluation.

RAG was integrated in experiments 2 and 4, providing the model with relevant external knowledge during inference.

Figure 9 presents a sequence diagram outlining the execution flow required for conducting the experiments described in this work. The diagram illustrates the interactions between key components, detailing the steps involved in data retrieval, processing and model inference.

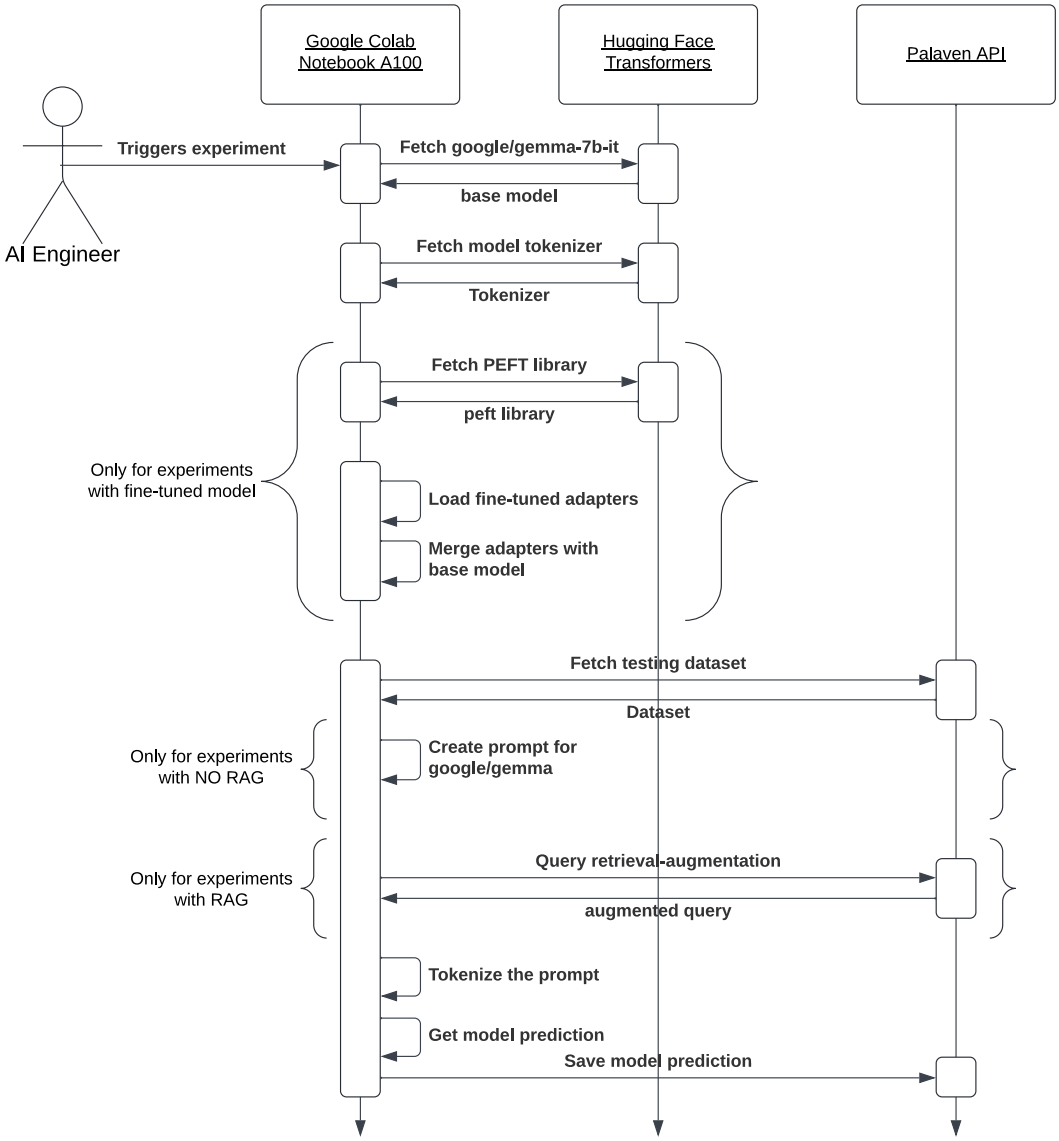

Figure 9. Experiments sequence diagram

4.4.1. Evaluation metrics

The following metrics were computed to assess the precision, recall and F1 score of the model responses:

- **BERTScore**: Measures semantic similarity between the generated response and the ground truth.

- **ROUGE (ROUGE-1, ROUGE-2, ROUGE-L)**: Evaluates n-gram overlap and summarization quality.
- **BLEU**: Assesses the fluency and correctness of the generated text based on n-gram precision.

Figure 10 illustrates the sequence diagram for the evaluation computing process. The diagram details the step-by-step interactions between components involved in evaluating model performance, metric computation, and result storage.

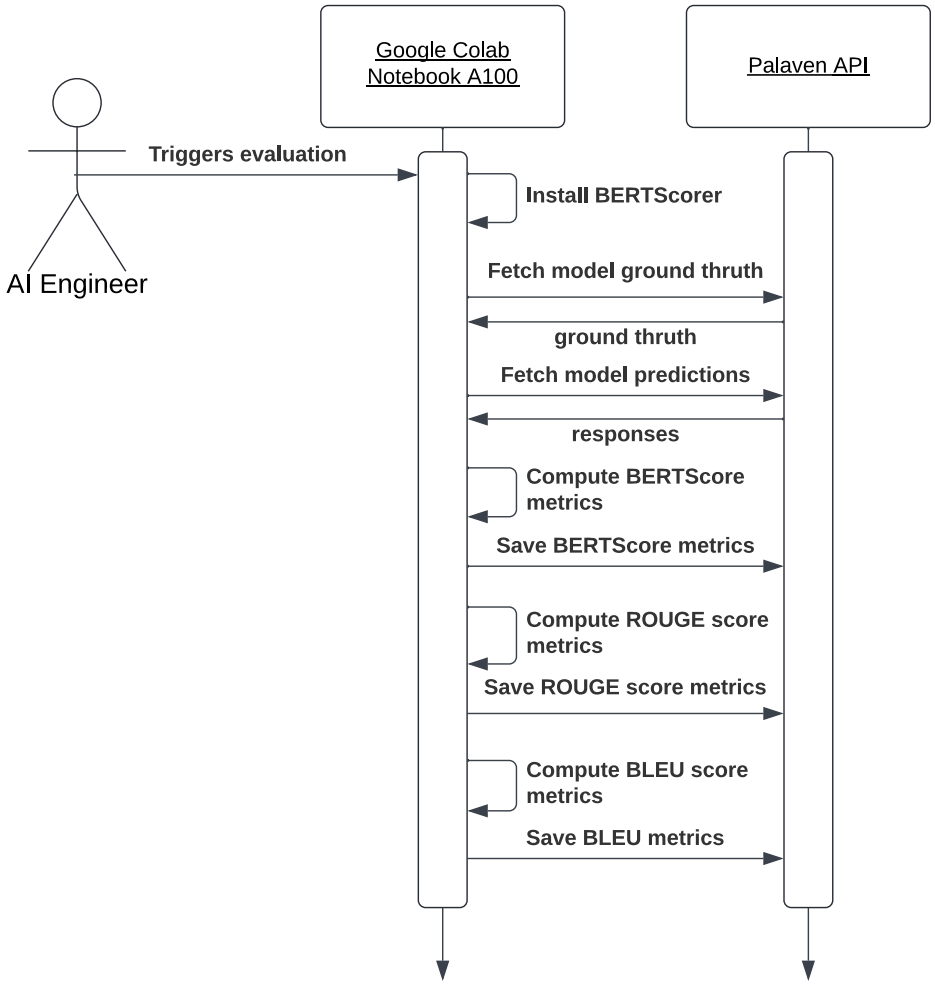

Figure 10. Evaluation computing sequence diagram

4.4.2. Fine-tuning Parameters

The fine-tuning process was executed on a notebook equipped with an A100 GPU with 40GB of memory. This hardware limitation dictated the selection of training parameters, as the model's size and the memory requirements for efficient training constrained the allowable batch sizes and sequence lengths. While

alternative configurations might yield improved results, they would likely require more substantial hardware resources.

The following are the key parameters utilized during the fine-tuning process, along with a brief explanation of their purpose and justification given the hardware constraints:

4.4.2.1. Batch Sizes

Parameter	Value	Description
per_device_train_batch_size	4	This parameter controls the number of training examples processed in a single forward and backward pass. A small batch size of 4 was chosen to fit within the memory limitations of the A100 GPU. Larger batch sizes would have exceeded the available memory, leading to out-of-memory (OOM) errors [48][49][50][64].
per_device_eval_batch_size	1	During evaluation, the batch size was set to 1 to minimize memory usage, ensuring that the model could still fit within the GPU during inference [48][49][50][64].

4.4.2.2. Gradient Accumulation

Parameter	Value	Description
gradient_accumulation_steps	4	To simulate a larger effective batch size without overloading the GPU memory, gradient accumulation was set to 4. This means gradients are accumulated over 4 steps before performing a backpropagation, effectively allowing for larger batch sizes [48][49][50][64].

4.4.2.3. Learning Rate and Optimization

Parameter	Value	Description
learning_rate	2e-4	The learning rate was set to a conservative value to ensure stable fine-tuning, especially given the large size of the google/gemma-7b-it model [48][49][50][64].
optim	paged_adamw_32bit	The AdamW optimizer was used in its 32-bit mode to balance performance and memory usage, helping to manage the size of gradients and weight updates [48][49][50][64].

4.4.2.4. Sequence Length

Parameter	Value	Description
max_seq_length	1459	The maximum sequence length was set to 1,459 tokens, which corresponds to the longest input sequences that could fit within the GPU memory. Sequence lengths beyond this would

	significantly increase the memory footprint, leading to OOM errors [48][49][50][64].

4.4.2.5. Mixed Precision

Parameter	Value	Description
bf16	true	Mixed precision training using bfloat16 (bf16) was enabled to reduce memory usage. This allows the model to store tensors using 16-bit precision while maintaining sufficient accuracy for the fine-tuning task. The use of bf16, rather than fp16, leverages the A100 GPU's native support for bfloat16 [48][49][50][64].

4.4.2.6. LoRA Fine-Tuning Parameters

Parameter	Value	Description
lora_r	8	This parameter controls the rank of the low-rank approximation matrices used by LoRA. A value of 8 was chosen to strike a balance between model capacity and memory efficiency [48][49][50][64].
lora_alpha	16	The scaling factor for the LoRA update matrices was set to 16, which ensures that the updates during fine-tuning are scaled appropriately without overwhelming the base model [48][49][50][64].
lora_dropout	0.1	Dropout was applied to the LoRA layers to prevent overfitting during fine-tuning. A 10% dropout rate was selected as a standard regularization technique [48][49][50][64].

4.4.2.7. Gradient Checkpointing

Parameter	Value	Description
gradient_checkpointing	true	Gradient checkpointing was enabled to reduce memory usage during backpropagation. By storing intermediate activations selectively, this technique allowed the training of large models within the GPU’s memory constraints, at the cost of additional computational overhead [48][49][50][64].

4.4.2.8. Training Steps and Warmup

Parameter	Value	Description
max_steps	1000	The model was trained for 1,000 steps, balancing between the computational budget and the need for sufficient training epochs to achieve effective fine-tuning [48][49][50][64].
warmup_ratio	0.03	A 3% warmup ratio was applied to the learning rate, gradually increasing it at the start of training to stabilize model convergence [48][49][50][64].

4.4.2.9. Logging and Evaluation

Parameter	Value	Description
logging_steps	10	Logs were recorded every 10 steps to closely monitor the model's performance and training dynamics [48][49][50][64].
eval_steps	100	The model was evaluated every 100 steps to try to reduce the overfitting [48][49][50][64].
save_steps	100	Checkpoints were saved every 100 steps to preserve model states throughout training, allowing for recovery in case of interruptions [48][49][50][64].

4.4.2.10. Quantization Configuration

Parameter	Value	Description
bnb_4bit_quant_type	nf4	To further optimize memory usage, 4-bit quantization was applied using the nf4 format. This reduced the model size and allowed efficient training while preserving model accuracy [48][49][50][64].

4.5. Source code and Technical Stack

The source code used to implement and evaluate the proposed methods is publicly available in the GitHub repository Palaven-LLM. This repository contains all the scripts and configurations needed to reproduce the experiments. Code available in: https://github.com/erickbr15/palaven-llm.

Table 1 presents the technical stack used in this work, detailing the key frameworks, libraries, and tools employed for data processing, model training, and deployment.

Technology /Library	Purpose
Python 3	Used to code the experiments and perform fine-tuning in Google Colab.
.NET/C#	Used for implementing backend business logic.
pandas	Employed for data manipulation and processing tasks.
numpy	Utilized for numerical computations and handling arrays.
matplotlib	Used for data visualization and plotting results.
sklearn	Provides tools for model evaluation and preprocessing.
pytorch	Serves as the primary deep learning framework for model training.
Google Colab	A cloud-based platform used for running experiments and training models.
Hugging Face Transformers	Used for working with pre-trained language models.
Hugging Face Supervised Trainer	Employed to set up the training arguments and create the supervised trainer for the LLM.
Hugging Face Training Utilities	Employed to use tools to optimize the training process.
Hugging Face LoRA Adapter	Employed to configure the LoRA adapter.
Hugging Face PEFT	Employed to merge the fine-tuned adapters with the original LLM.
BERTScore	Employed to evaluate the generated inferences using the BERTScore metric.

ROUGE	Employed to evaluate the generated inferences using the ROUGE 1,2,L metrics.
BLEU	Employed to evaluate the generated inferences using the BLEU metric.
Azure AI Document Intelligence	Employed for document processing tasks during the initial ETL steps.
Azure Functions	Employed to implement serverless host processes for the ETL and create semantic search index.
Azure Cosmos DB NoSQL API	Employed to store the medallion stages (bronze, silver, and golden documents) in JSON format.
Azure SQL DB	Employed to store evaluation session data, the synthetic instruction dataset, the synthetic fine-tuning dataset, the evaluation metric results and the fine-tuning batch progress.
ASP.NET Core Web API	Employed to implement utility API to address the evaluation session set up and manage the training/evaluation instruction batches.
Swagger	Provides helpers to expose API documentation following the standard OpenAPI.
Pinecone	Employed to create the semantic search index storing embedding vectors created using text-embedding-ada-002.
OpenAI GPT-3.5-Turbo API	Employed for extraction, summarization and text generation tasks during the ETL process.
OpenAI Embeddings API	Employed to create embedding vectors from text.

Table 1. Technical stack

5. RESULTS AND DISCUSSION

Abstract: *This chapter presents the results obtained during the fine-tuning process as well as the metrics obtained from measuring the performance of the LLM google/gemma-7b-it in the four experiments using BERTScore, ROUGE (1,2, L) and BLEU.*

5.1. Results

5.1.1. Fine-tuning results

The fine-tuning process was conducted using the base model google/gemma-7b-it, specifically adjusted for the task by employing PEFT with QLoRA. During training, a dataset of pre-defined instructions and responses was processed in a GPU environment (A100 with 40GB of memory). The model was optimized using gradient accumulation and mixed precision techniques (bf16) to manage the GPU's capacity. Additionally, RAG was incorporated in some experiments, allowing the model to access external information to enhance the accuracy and relevance of its responses.

5.1.1.1. Training Loss

This graph shows the evolution of training loss over the training steps. A continuous decrease in training loss indicates that the model is learning to minimize prediction error during training [48][49][50][64].

Figure 11 presents the fine-tuning training loss chart, illustrating the loss reduction over successive training iterations.

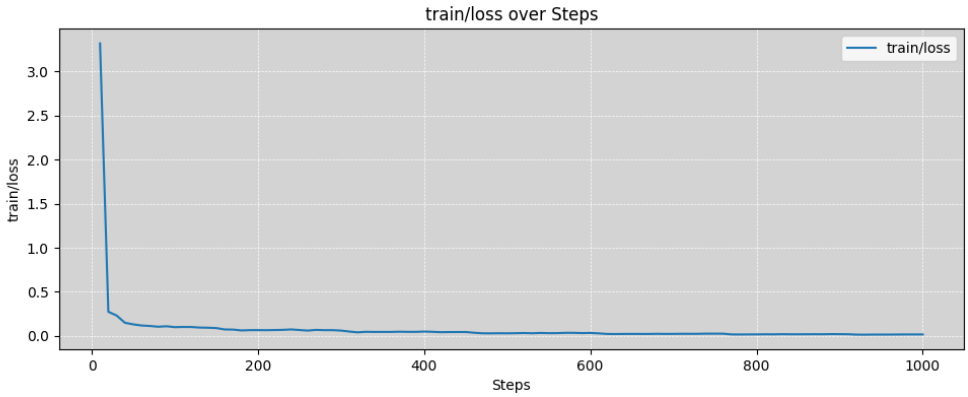

Figure 11. Fine-tuning training loss chart

5.1.1.2. Gradient Norm

The gradient norm graph monitors the magnitude of the gradients during training. The gradient norm reflects the size of the updates applied to the model's weights at each step. This helps to monitor that gradients do not become too large (which could cause instability) or too small (which could lead to learning stagnation) [48][49][50][64].

Figure 12 presents the fine-tuning gradient norm chart, depicting the magnitude of gradient updates across training iterations.

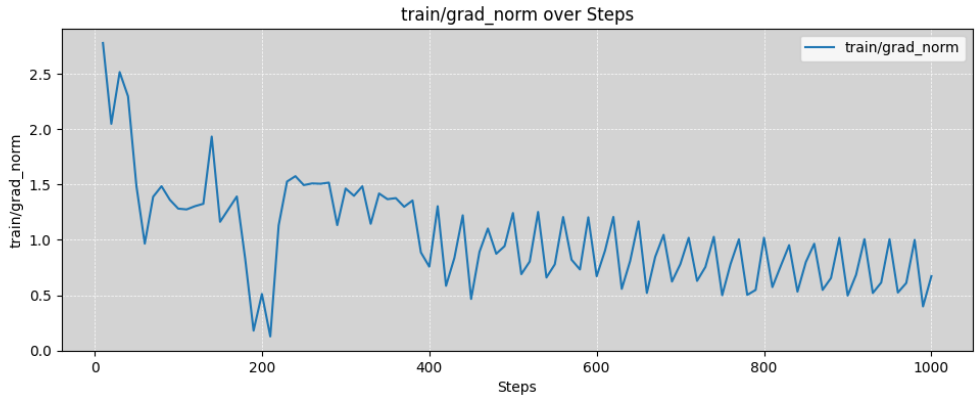

Figure 12. Fine-tuning gradient norm chart

5.1.1.3. Evaluation Loss

The evaluation loss measures the error in the model's predictions on a validation set that was not used during training. This metric is key for observing whether the model is generalizing correctly and not overfitting to the training set. An increase in evaluation loss may suggest overfitting, while a downward trend indicates improved model generalization [48][49][50][64].

Figure 13 presents the fine-tuning evaluation loss chart, illustrating the loss computed on the validation dataset across training iterations.

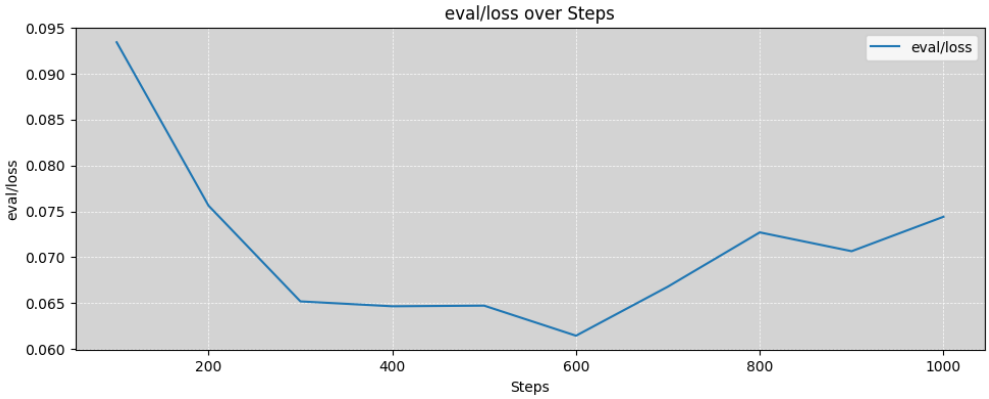

Figure 13. Fine-tuning evaluation loss chart

5.1.1.4. Evaluation Runtime

The evaluation runtime graph shows how long the model takes to complete each evaluation. This metric is useful for understanding the model's efficiency during the evaluation process and for optimizing inference in production environments. A lower evaluation runtime indicates a more efficient model, which is critical for real-time or time-constrained applications [48][49][50][64].

Figure 14 presents the fine-tuning evaluation runtime chart, illustrating the time required to compute evaluation metrics across training iterations. This chart provides insights into the computational efficiency of the fine-tuning process, helping to assess the impact of model size, batch processing, and hardware utilization.

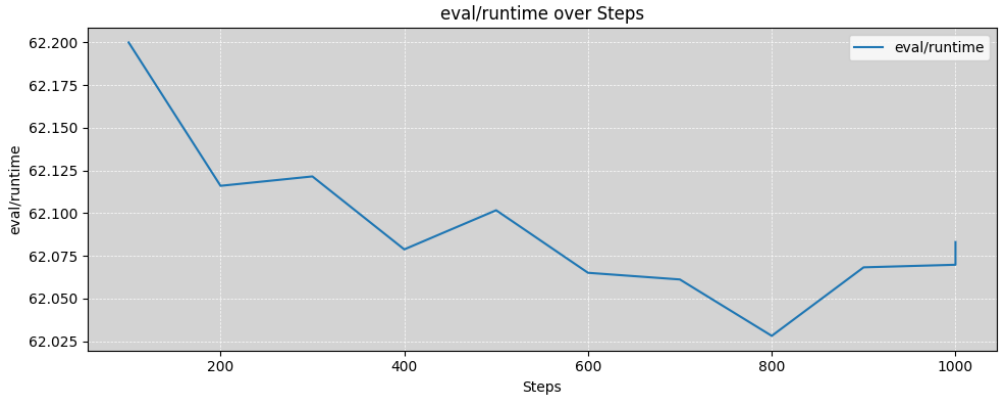

Figure 14. Fine-tuning evaluation runtime chart

5.1.2. BERTScore evaluation results

BERTScore measures the semantic similarity between the generated text and the reference ground truth. It evaluates Precision, Recall, and F1-scores for each batch of generated instructions. The fine-tuned model, particularly the one using RAG, demonstrated higher BERTScore values compared to the other configurations, indicating better semantic alignment with the reference text [22][53].

Table 2 presents the BERTScore evaluation metrics, including precision, recall, and F1 scores for the four experiments.

Experiment #	Experiment	Precision	Recall	F1
1	llmvanilla	0.66329636	0.73575428	0.69674371
2	llmrag	0.71945688	0.82890935	0.76927271
3	llmfinetuned	0.75361674	0.78112177	0.76609989
4	llmfinetunedrag	0.78605901	0.84935553	0.81492597

Table 2. BERTScore evaluation metrics

Figure 15 presents the BERTScore evaluation metrics chart, comparing the results obtained from the four experiments. The chart highlights performance differences across experiments, confirming the hypothesis that the fine-tuned LLM + RAG approach achieved the best performance relative to the other configurations.

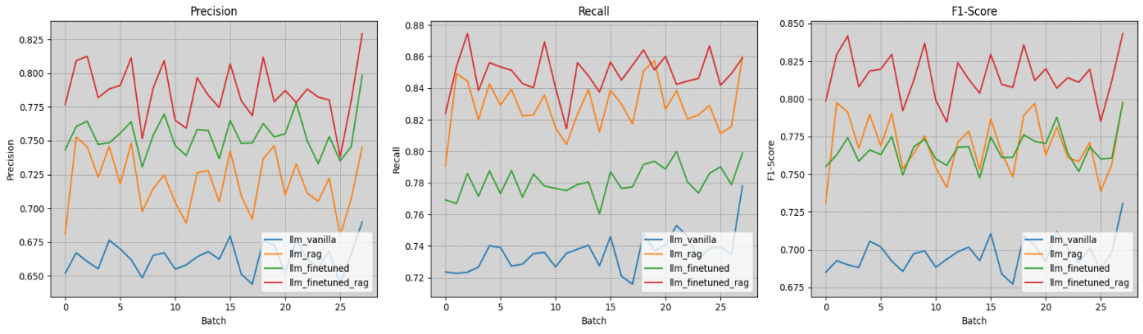

Figure 15. BERTScore evaluation metrics chart

5.1.3. ROUGE (1,2-L) evaluation results

The evaluation of the experiments using ROUGE metrics—ROUGE-1, ROUGE-2, and ROUGE-L—demonstrated that the highest values were consistently achieved in the experiments that incorporated RAG. Among the four experiments, the combination of fine-tuning and RAG produced the best overall results.

ROUGE metrics are designed to measure the overlap between the generated text and the ground truth, providing insights into how closely the model's output aligns with expected responses [23]. These metrics are widely used in text summarization and natural language generation tasks due to their ability to assess content similarity effectively:

- ROUGE-1 evaluates unigram (word-level) overlap, providing a basic measure of content similarity.
- ROUGE-2 measures bigram (two-word sequence) overlap, capturing more context than individual words.
- ROUGE-L assesses the longest common subsequence, which reflects how well the model captures the overall structure and coherence of the text [23].

By utilizing these evaluation metrics, the study validates the impact of incorporating retrieval-augmented generation in improving the accuracy and coherence of the model's output..

Table 3 presents the ROUGE evaluation metrics, including ROUGE-1, ROUGE-2, and ROUGE-L scores for the four experiments.

Experiment #	Experiment	ROUGE type	Precision	Recall	F1
1	llmvanilla	ROUGE-1	0.19607324	0.56276667	0.26535556
2	llmrag	ROUGE-1	0.30373967	0.8289285	0.41644026
3	llmfinetuned	ROUGE-1	0.39414989	0.53591594	0.42689081
4	llmfinetunedrag	ROUGE-1	0.48617067	0.76572811	0.55496444
1	llmvanilla	ROUGE-2	0.09587356	0.26615819	0.12800334
2	llmrag	ROUGE-2	0.23128762	0.61633446	0.31561576
3	llmfinetuned	ROUGE-2	0.234449	0.3096749	0.25217805
4	llmfinetunedrag	ROUGE-2	0.37903237	0.59135093	0.4342105
1	llmvanilla	ROUGE-L	0.14562976	0.43349436	0.19869995
2	llmrag	ROUGE-L	0.26249351	0.72223404	0.36023419
3	llmfinetuned	ROUGE-L	0.31749912	0.43508286	0.3437339
4	llmfinetunedrag	ROUGE-L	0.4335472	0.68773563	0.49608514

Table 3. ROUGE evaluation metrics

Figure 16 presents the ROUGE evaluation metrics chart, comparing the results obtained from the four experiments. The chart displays ROUGE-1, ROUGE-2, and ROUGE-L scores, confirming the hypothesis that the fine-tuned LLM + RAG approach achieved the best performance among the evaluated configurations.

Figure 16. ROUGE evaluation metrics chart

5.1.4. BLEU score evaluation results

BLEU evaluates the fluency and grammatical correctness of the generated text based on n-gram precision. It is commonly used for tasks like machine translation but is also applicable to general text generation [24].

Table 4 presents the BLEU evaluation metrics for the four experiments, assessing the accuracy of generated text.

Experiment #	Experiment	BLEU Score
1	llmvanilla	0.053468905
2	llmrag	0.180205894
3	llmfinetuned	0.149816647
4	llmfinetunedrag	0.296071502

Table 4. BLEU evaluation metrics

Figure 17 presents the BLEU evaluation metrics chart, comparing the results obtained from the four experiments. The chart demonstrates that the fine-tuned LLM + RAG approach achieved the highest performance among the evaluated configurations.

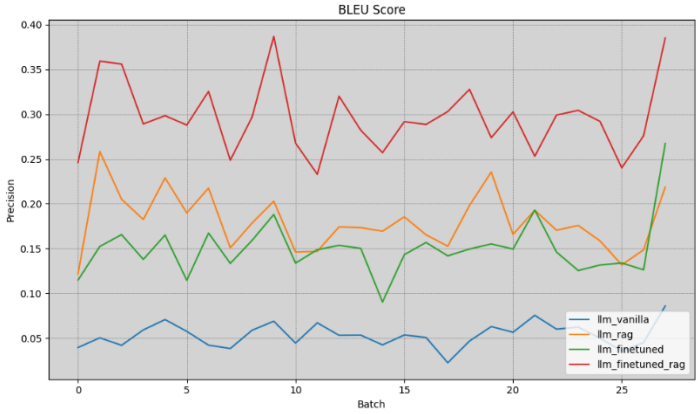

Figure 17. BLEU evaluation metrics chart

6. CONCLUSIONS

Abstract: *This chapter presents conclusions and future work regarding enhancing LLM performance in specialized Spanish domains using RAG and PEFT QLoRA.*

6.1. Conclusions

The experiments conducted in this work provide evidence supporting the hypothesis that combining a RAG module with a fine-tuned LLM via PEFT QLoRA significantly improves the model's performance on specialized knowledge domains. While the results validate the hypothesis, the evaluation also reveals the strengths and limitations of the metrics used.

N-gram metrics like BLEU and ROUGE provide insight into the overlap between generated and reference texts. While these metrics are valuable, they have limitations, particularly in their inability to account for deeper semantic understanding, as they primarily focus on surface-level word overlaps [23][24][63].

BERTScore leverages contextual embeddings to measure the semantic similarity between generated and reference texts, providing a more comprehensive evaluation than n-gram-based metrics. The BERTScore achieved in this work suggests that the model generates semantically relevant content. This score, in combination with the n-gram metrics, supports that the fine-tuning in addition of RAG is effective in both retrieving and generating relevant content [22][63].

These results suggest that the methodology used here—combining domain-specific fine-tuning with the retrieval of relevant external knowledge—is an effective strategy for improving the accuracy and relevance of LLM-generated responses.

On the other hand, the designed ETL (Extract, Transform, Load) process proved successful in structuring and preparing the data used for training the model. The architecture, based on Medallion principles, is both scalable and maintainable, ensuring that future iterations of the system can easily incorporate additional data sources with minimal reconfiguration. Importantly, this architecture is customizable and extendable to include the full Mexican legal framework available on the SAT's normativity page. This feature highlights the potential for expanding the system to address broader legal contexts, providing a foundation for ongoing improvements and expansion.

6.2. Future Work

Despite the positive outcomes achieved, there remain several opportunities to further refine and expand this work. The following are key areas for future exploration:

- Comprehensive Document Inclusion: While the current experiments have focused on a specific subset of legal documents, future work should extend this to include all legal documents relevant to the domain. Ensuring full coverage of the legal corpus will enhance the model's ability to retrieve and generate more accurate and comprehensive responses, particularly for complex legal queries.

- Human Validation in Document Creation: The creation of Silver and Golden documents in the ETL pipeline is critical to ensuring the quality and reliability of the data used for both training and retrieval. Future implementations should incorporate human validation processes to verify the accuracy and integrity of these documents. This would improve the overall reliability of the model's responses and help mitigate potential biases or errors introduced during data preparation.

- Agentic Reasoning with Mathematical and Tabular Data: Another avenue for improvement involves expanding the model's capability to recognize when additional external information, such as mathematical formulas or tabular data, is required to formulate an accurate response. Integrating an agentic reasoning approach, where the model can autonomously decide to query supplementary data sources, when necessary, would enhance its ability to handle complex queries that involve numerical or data-driven elements.

- Reinforcement Learning by Human Feedback (RLHF): Future iterations of the model could benefit from the incorporation of RLHF, allowing the model to continuously improve its response quality based on user feedback. This technique would help the model learn to generate more useful, contextually relevant, and accurate responses over time, adapting to the evolving needs of users [20].

- Optimization and Scalability for Production: For the deployment of the model in production environments, further research should focus on optimizing the inference process to ensure low latency and high throughput. Additionally, the development of a scalable and maintainable architecture for chatbot applications will be essential to support real-time interactions with end-users. This includes ensuring robust scalability to handle high traffic and maintaining the flexibility needed to incorporate updates and improvements to the model over time.

BIBLIOGRAPHY

[1] P. Lewis, E. Perez, A. Piktus, et al., "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks" arXiv preprint arXiv:2005.11401, May 2020. [Online]. Available: http://arxiv.org/abs/2005.11401

[2] V. Lialin, V. Deshpande, A. Rumshisky, "Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning," UMass Lowell and Alexa AI, arXiv preprint arXiv:2303.15647, Mar. 2023. [Online]. Available: https://arxiv.org/abs/2303.15647

[3] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "QLORA: Efficient Finetuning of Quantized LLMs," University of Washington, arXiv preprint arXiv:2305.14314, May 2023. [Online]. Available: https://arxiv.org/abs/2305.14314

[4] C. Ling, X. Zhao, J. Lu, C. Deng, C. Zheng, J. Wang, T. Chowdhury, Y. Li, H. Cui, X. Zhang, T. Zhao, A. Panalkar, D. Mehta, S. Pasquali, W. Cheng, H. Wang, Y. Liu, Z. Chen, H. Chen, C. White, Q. Gu, J. Pei, C. Yang, and L. Zhao, "Domain Specialization as the Key to Make Large Language Models Disruptive: A Comprehensive Survey," arXiv preprint arXiv:2305.18703, Oct. 2023. [Online]. Available: https://arxiv.org/abs/2305.18703

[5] O. Ovadia, M. Brief, M. Mishaeli, and O. Elisha, "Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs," Microsoft, Israel, arXiv preprint arXiv:2312.05934, Jan. 2024. [Online]. Available: https://arxiv.org/abs/2312.05934

[6] SAT, "Normatividad," [Online]. Available: https://www.sat.gob.mx/personas/normatividad.

[7] Ng, Generative AI for Everyone, DeepLearning.AI, DeepLearning.AI, Nov. 2023. [Online]. Available: https://www.deeplearning.ai/courses/generative-ai-for-everyone/.

[8] Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Proc. Advances in Neural Information Processing Systems (NeurIPS), vol. 30, 2017.

[9] J. Alammar and M. Grootendorst, Hands-On Large Language Models. O'Reilly Media, Inc., 2024.

[10] Fulford and A. Ng, ChatGPT Prompt Engineering for Developers, OpenAI and DeepLearning.AI, DeepLearning.AI, Jan. 2024. [Online]. Available: https://www.deeplearning.ai/short-courses/chatgpt-prompt-engineering-for-developers/.

[11] Prompt Engineering Guide. [Online]. Available: https://www.promptingguide.ai/.

[12] What is the medallion lakehouse architecture?, Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/azure/databricks/lakehouse/medallion.

[13] Azure Data Factory on Azure landing zones baseline architecture, Microsoft Learn. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/databases/architecture/azure-data-factory-on-azure-landing-zones-baseline.

[14] N. Muennighoff, N. Tazi, L. Magne, and N. Reimers, "MTEB: Massive Text Embedding Benchmark," arXiv, vol. 2210.07316, Oct. 2022.

[15] Fulford and A. Ng, Building Systems with the ChatGPT API, OpenAI and DeepLearning.AI, DeepLearning.AI, Jan. 2024. [Online]. Available: https://www.deeplearning.ai/short-courses/building-systems-with-chatgpt/.

[16] Retrieval Augmented Generation (RAG) for LLMs. [Online]. Available: https://www.promptingguide.ai/research/rag.

[17] J. Liu, Building Agentic RAG with LlamaIndex, LlamaIndex and DeepLearning.AI, DeepLearning.AI, May 2024. [Online]. Available: https://www.deeplearning.ai/short-courses/building-agentic-rag-with-llamaindex/.

[18] S. Witalec, Vector Databases: from Embeddings to Applications, Weaviate and DeepLearning.AI, DeepLearning.AI, May 2024. [Online]. Available: https://www.deeplearning.ai/short-courses/vector-databases-embeddings-applications/.

[19] Y. Belkada and M. Sun, Quantization Fundamentals with Hugging Face, Hugging Face and DeepLearning.AI, DeepLearning.AI, June 2024. [Online]. Available: https://www.deeplearning.ai/short-courses/quantization-fundamentals-with-hugging-face/.

[20] M. Chambers, A. Barth, C. Fregly, and S. Eigenbrode, Learn the Fundamentals of Generative AI for Real-World Applications, AWS and DeepLearning.AI, DeepLearning.AI, Jan. 2024. [Online]. Available: https://www.deeplearning.ai/courses/generative-ai-with-llms/.

[21] Flanagan and D. Timofeev, TECH16 Large Language Models for Business with Python, Stanford, Stanford Online, Feb.-Apr. 2024.

[22] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi, "BERTScore: Evaluating Text Generation with BERT," in Proc. International Conference on Learning Representations (ICLR), 2020. [Online]. Available: https://arxiv.org/abs/1904.09675.

[23] Y. Lin, "ROUGE: A Package for Automatic Evaluation of Summaries," in Text Summarization Branches Out, 2004. [Online]. Available: https://aclanthology.org/W04-1013/.

[24] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "BLEU: a method for automatic evaluation of machine translation," in Proc. 40th Annual Meeting of the Association for Computational Linguistics, Philadelphia, PA, USA, 2002, pp. 311–318.

[25] OpenAI Platform - Embedding models. [Online]. Available: https://platform.openai.com/docs/guides/embeddings/embedding-models.

[26] New and improved embedding model, OpenAI, Dec. 2022. [Online]. Available: https://openai.com/index/new-and-improved-embedding-model/.

[27] OpenAI, "Embeddings." [Online]. Available: https://platform.openai.com/docs/guides/embeddings.

[28] OpenAI, "GPT-3.5 Turbo." [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5-turbo.

[29] OpenAI, "Chat completion." [Online]. Available: https://platform.openai.com/docs/guides/chat-completions.

[30] Models, Gemma. [Online]. Available: https://www.promptingguide.ai/models/gemma.

[31] google/gemma-7b-it model card, Hugging Face. [Online]. Available: https://huggingface.co/google/gemma-7b-it.

[32] google/gemma-7b-it, Kaggle. [Online]. Available: https://www.kaggle.com/models/google/gemma/Transformers/7b-it/1.

[33] NET documentation. [Online]. Available: https://learn.microsoft.com/en-us/dotnet/.

[34] Azure Architecture Reference, ETL. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/data-guide/relational-data/etl.

[35] Azure Cosmos DB for NoSQL documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/cosmos-db/nosql/.

[36] Azure Blob Storage documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/storage/blobs/.

[37] Azure Functions documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/.

[38] Azure AI Document Intelligence documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/ai-services/document-intelligence/?view=doc-intel-4.0.0.

[39] Pinecone API reference. [Online]. Available: https://docs.pinecone.io/reference/api/introduction.

[40] Google Colab. [Online]. Available: https://colab.research.google.com/.

[41] Python documentation. [Online]. Available: https://docs.python.org/3/.

[42] Pytorch. [Online]. Available: https://pytorch.org/docs/stable/index.html.

[43] How to Fine-Tune an LLM Part 1: Preparing a Dataset for Instruction Tuning. [Online]. Available: https://wandb.ai/capecape/alpaca_ft/reports/How-to-Fine-Tune-an-LLM-Part-1-Preparing-a-Dataset-for-Instruction-Tuning--Vmlldzo1NTcxNzE2.

[44] pandas library. [Online]. Available: https://pandas.pydata.org/.

[45] sklearn.model_selection. [Online]. Available: https://scikit-learn.org/0.19/modules/classes.html#module-sklearn.model_selection.

[46] How to Fine-tune an LLM Part 3: The HuggingFace Trainer. [Online]. Available: https://wandb.ai/capecape/alpaca_ft/reports/How-to-Fine-tune-an-LLM-Part-3-The-HuggingFace-Trainer--Vmlldzo1OTEyNjMy.

[47] Hugging Face Transformers. [Online]. Available: https://huggingface.co/docs/transformers/index.

[48] Supervised Fine-tuning Trainer. [Online]. Available: https://huggingface.co/docs/trl/v0.9.6/en/sft_trainer.

[49] Utilities for Trainer. [Online]. Available: https://huggingface.co/docs/transformers/internal/trainer_utils.

[50] LoRA adapter. [Online]. Available: https://huggingface.co/docs/peft/package_reference/lora.

[51] Making LLMs even more accessible with bitsandbytes, 4-bit quantization and QLoRA. [Online]. Available: https://huggingface.co/blog/4bit-transformers-bitsandbytes.

[52] PEFT library. [Online]. Available: https://pypi.org/project/peft/0.7.0/.

[53] Bertscore library. [Online]. Available: https://pypi.org/project/bert-score/.

[54] ROUGE library. [Online]. Available: https://pypi.org/project/rouge/.

[55] BLEU library. [Online]. Available: https://pypi.org/project/bleu/.

[56] Azure SQL documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-sql/?view=azuresql.

[57] J. P. Hendrycks, S. Basart, M. Mazeika, A. Zou, M. Chen, and D. Song, "Measuring Massive Multitask Language Understanding," in Proc. International Conference on Learning Representations (ICLR), 2021. [Online]. Available: https://arxiv.org/abs/2009.03300

[58] Create web APIs with ASP.NET Core. [Online]. Available: https://learn.microsoft.com/en-us/aspnet/core/web-api/?view=aspnetcore-8.0&WT.mc_id=dotnet-35129-website.

[59] API Management documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/api-management/.

[60] App Service documentation. [Online]. Available: https://learn.microsoft.com/en-us/azure/app-service/.

[61] Matplotlib documentation. [Online]. Available: https://matplotlib.org/stable/index.html.

[62] Numpy. [Online]. Available: https://numpy.org/.

[63] RAG evaluation metrics: A journey through metrics. [Online]. Available: https://www.elastic.co/search-labs/blog/evaluating-rag-metrics.

[64] Hugging Face, Methods and tools for efficient training on a single GPU. [Online]. Available: https://huggingface.co/docs/transformers/perf_train_gpu_one.