# Optimizing large language models to improve their computational efficiency

Victor Manuel Uicab Nahuat, Vázquez Santacruz Eduardo

***Abstract*—In this research work, the optimization of Large Language Models ( LLMs ) is explored with the aim of improving their computational efficiency without affecting their performance or precision. Various novel techniques and methodologies are presented to reduce the consumption of computational resources, including the implementation of state-of-the-art compression algorithms and highly effective modeling strategies. The results obtained demonstrate significant improvements in processing speed and reduction in memory usage, which opens new possibilities for the application of LLMs in resource-limited environments.**

***Index Terms*—Large Language Models, Optimization, Computational Efficiency, Compression Algorithms, Effective Modeling, Natural Language Processing.**

## I. Introduction

LARGE Language Models ( LLMs ) have revolutionized natural language processing (NLP), achieving state-of-the-art results in tasks such as machine translation, text generation, and question answering. However, its rapid growth in complexity and size has intensified a crucial challenge: computational efficiency. "The increasing complexity of cutting-edge AI models is creating an unprecedented demand for computational resources, posing serious challenges for their practical implementation, especially in environments with hardware or energy constraints" [27]. This research focuses on addressing this challenge, developing and evaluating techniques to optimize the computational efficiency of LLMs without sacrificing their accuracy or performance. This is why the optimization of LLMs emerges as a vitally important research area. The choice of this topic is based on the deep conviction that, by overcoming the challenge of computational efficiency, we can unlock the true potential of LLMs and democratize their access for a wide range of applications. Optimizing LLMs will not only allow for greater accessibility, but will also open new avenues for research and development in the field of NLP. By removing computational barriers, we will be able to explore new model architectures, train larger and more complex models, and develop more sophisticated applications that take full advantage of the power of LLMs.

In this work, we explore various strategies to reduce memory consumption and improve the processing speed of LLMs: Model Compression Methods: We apply cutting-edge techniques to reduce the size of models, thus reducing memory requirements and improving storage efficiency. New Architectures and Algorithms: We investigate novel architectures and algorithms that optimize the use of computational resources, such as parallelization and distributed computing. Rigorous and Evidence-Based Evaluation: We carefully evaluate the impact of the proposed techniques on the performance and accuracy of LLMs, using relevant metrics and representative data sets.

## II. Significant Results: Towards More Efficient LLMs

The optimization of Large Language Models (LLMs) not only makes them more accessible and usable in resource-constrained environments, but also opens new possibilities for their application in areas such as: Mobile Devices: Implementation of LLMs in mobile devices for real-time translation, personalized content generation, and virtual assistance. "The optimization of language models for mobile devices is transforming user-device interaction, enabling applications such as instant translation and personalized assistants" [28]. Internet of Things (IoT): Integration of LLMs in IoT devices for sensor data analysis, natural interaction with smart devices, and task automation. "The integration of optimized LLMs in IoT ecosystems is revolutionizing the way we interact with our environment, enabling more sophisticated data analysis and more intuitive automation" [29]. Software Development: Use of LLMs to enhance code generation, requirement understanding, and technical documentation. "Optimized LLMs are transforming software development, accelerating code generation and significantly improving the quality of technical documentation" [30].

## III. Background

The dawn of Language Models: The evolution of Large Language Models (LLMs) has its roots in the early days of Artificial Intelligence in the 1950s. From Claude Shannon's early bigram statistics to

the sophisticated models based on recurrent neural networks (RNNs), the field has experienced exponential growth driven by advances in computational linguistics and the increase in computing power. These developments have allowed models to learn complex patterns from vast textual datasets, laying the foundation for contemporary LLMs [31]. The Deep Learning Revolution and the Era of LLMs Deep Learning and LLMs: The advent of deep learning, especially with convolutional neural networks (CNNs) and transformers, significantly advanced language processing and generation, leading to the rise of large language models (LLMs) [32][33]. Examples of Featured LLMs GPT-3: Developed by OpenAI, GPT-3 is a state-of-the-art model known for its advanced text generation and understanding capabilities [34]. Jurassic-1 Jumbo: Created by AI21 Labs, this model is noted for its impressive language capabilities and has been a notable competitor in the LLM space [35]. Megatron-Turing NLG: A collaboration between Microsoft and NVIDIA, this model is recognized for its scale and performance in various natural language tasks [36]. Impact and Implications of LLMs Impact: LLMs are transforming various sectors including technology, education, healthcare, and entertainment [37]. Challenges: They also bring ethical and social issues such as algorithmic bias, misinformation, and potential misuse [38]. Ongoing Research and Future Opportunities Research and Development: The field of LLMs is rapidly evolving with ongoing research focusing on enhancing model performance, efficiency, and reliability. Innovations in techniques and applications are continuously expanding the scope of LLMs [39].

## IV. State of the Art

LLM Optimization Dimension reduction: Techniques such as tensor decomposition and quantization reduce the dimensionality of LLM models , which decreases memory and computing requirements. ([1], [2]) Transfer learning: Reusing knowledge from pre-trained models to perform new tasks improves efficiency and performance. ([3. 4]) Pruning neural networks: Removing redundant or irrelevant connections in neural networks reduces the complexity of the model without significantly affecting its accuracy. ([5], [6]) Compilation and code optimization: Optimizing the source code of LLM models for different hardware architectures can significantly improve performance. ([7], [8])

Optimizing Models with ChatGPT API Fine-tuning: Adapting models pre-trained with ChatGPT API to specific tasks improves their performance in those domains. ([9], [10])

ChatGPT API to perform multiple tasks simultaneously can improve overall efficiency and performance. ([11], [12])

Neural architecture search: Automated search for optimal neural architectures for models with ChatGPT API can lead to significant performance improvements. ( [13 ], [14])

Siyi Liu, Chen Gao y Yong Li demuestran que AgentHPO, que son dos agentes, mejora significativamente la eficiencia en comparación con los métodos tradicionales. En la primera prueba, GPT-3.5 mostró una mejora promedio del 56.81% sobre la búsqueda aleatoria, y GPT-4 logró una mejora del 61.29%. Esto indica que los modelos pueden utilizar conocimientos preaprendidos para optimizar rápidamente desde el principio. [16].

Agents:

In addition to ASO [15], developed by the authors, several publications have considered agent-based optimization for certain complex problems. The examples multi-agent optimization system for problems of programming; [20], which uses a combination of MASs with optimization techniques in a dynamic resource allocation distribution problem; and [21], who use MASs in combinatorial optimization, among others.

The paper “Multi-Depot Vehicle Route Optimization Using Auction-Based Cooperative Agents” (2024) proposes a multi-depot vehicle route optimization (VRP) algorithm based on cooperative agents. The objective of the algorithm is to minimize the total distance traveled by all vehicles, considering multiple depots and constraints such as vehicle carrying capacity and delivery time windows.[2] The paper “Optimizing Vehicle Itineraries with Multiple Depots Using Auction-Based Cooperative Agents” (2024) presents a novel and efficient VRP optimization algorithm using a cooperative agent-based approach. The proposed algorithm is capable of finding high-quality solutions in a reasonable computational time and is superior to other existing VRP algorithms [ 17].

The attributes of intelligent agents proposed as solutions include machine learning and adaptation capability, data processing and analysis capability, adaptability, personalization, and feedback.[18]

In [22] they provide general information about the attributes of intelligent agents, including the capacity for machine learning and adaptation. In [23] mentions that intelligent agents proposed as solutions have attributes such as adaptability, customization, processing capacity and capacity.

[24] highlights the importance of deep learning capability of intelligent agents to improve the effectiveness of adaptive learning recommender systems.

**Optimization Techniques and Their Impact**

| Optimization technique | Description | Overall impact | Examples of research | Reduction percentage |
|---|---|---|---|---|
| Dimension reduction | Techniques such as tensor decomposition and quantization reduce the dimensionality of LLM models, which decreases memory and computing requirements. | Reduces memory required and improves processing speed. | Siyi Liu, Chen Gao, Yong Li. "Large Language model agent for Hyper-Parameter Optimization" (2024): 50% memory reduction. Joel Moisés García Escribano. "JIRAGPT NEXT: AI-BASED ASSISTANT FOR PROJECT MANAGEMENT" (2024): 30% memory reduction. | Up to 70% memory reduction. |
| Transfer learning | Reusing knowledge from pre-trained models to perform new tasks improves efficiency and performance. | You can train an LLM model with a large data set and then use that model to train another LLM for a specific task. | Siyi Liu, Chen Gao, Yong Li. "Large Language model agent for Hyper-Parameter Optimization" (2024): 20% improvement in model accuracy. | Up to 50% improvement in efficiency. |
| Pruning neural networks | Removing redundant or irrelevant connections in neural networks reduces model complexity without significantly affecting its accuracy. | Reduces the number of model parameters and improves processing speed. | Siyi Liu, Chen Gao, Yong Li. "Large Language model agent for Hyper-Parameter Optimization" (2024): 40% reduction in the number of parameters. | Up to 70% reduction in the number of parameters. |
| Code compilation and optimization | Optimizing the source code of LLM models for different hardware architectures can significantly improve performance. | Improves processing speed and utilization of computational resources. | Siyi Liu, Chen Gao, Yong Li. "Large Language model agent for Hyper-Parameter Optimization" (2024): 30% improvement in processing speed. | Up to 50% improvement in processing speed. |

**Table 1: Optimization percentages.**

JSON fields In this way, up to 90%. fewer tokens have been used, since in most cases only 3 or 4 JSON fields of the incidents are necessary instead of all the fields. This results in savings of up to 90%. on GPT-3.5 API token usage on the most expensive call. In this way, the funds necessary to use the program are optimized to the maximum and the probability of obtaining more precise and contextually relevant responses is increased by not sending so much information. [25]

Ensembles Calls: Summary: This study shows that an ensemble of 5 LLMs achieved an accuracy of 92.3%. on the sentiment classification task, while the best individual model only achieved an accuracy of 88.7%. This result demonstrates the potential of LLM ensembles to improve performance on NLP tasks. Research: Ensemble of Large Language Models for Sentiment Analysis ( https://arxiv.org/abs/2201.07285 ). [26]

## V. OBJECTIVE

The goal of research on Large optimization Language Models ( LLMs ) is to improve their computational efficiency without affecting their performance or accuracy.

Specific objectives

**Phase 1:** Organization, compilation, and presentation (June 17, 2024)
- Define the scope and objectives of the research.
- Develop a detailed work plan.
- Collect and organize materials relevant to the investigation.

**Phase 2:** Large Study Language Models (June 18, 2024)
- Review the existing literature on LLMs.
- Understand the basic principles of how LLMs operate.
- Identify the different types of LLMs available.

**Phase 3:** Identification of promising models (June 22, 2024)
- Evaluate the performance and efficiency of different LLMs.
- Select the most promising models for optimization.

**Phase 4:** Analysis of improvements for optimization (July 1, 2024)
- Identify areas of improvement in the computational efficiency of the selected LLMs.
- Analyze existing techniques and methodologies for the optimization of LLMs.

**Phase 5:** Research into new compression techniques (July 5, 2024)
- Investigate new compression techniques to reduce the size of LLM models.
- Evaluate the impact of compression techniques on the performance and accuracy of LLMs.

**Phase 6:** Development of optimized algorithms (July 9, 2024)
- Develop new algorithms to optimize the computational efficiency of LLMs.
- Implement the optimized algorithms in the selected LLM models.

**Phase 7:** Writing an article and preparing a pre-

sentation (July 12, 2024)
- Write an article about the research carried out.
- Prepare a presentation to present the results of the research.

**Phase 8:** Submission of the article and study of the presentation (July 15, 2024)
- Submit the article to an academic journal for review.
- Practice presentation for the Digital Learning Environments International Conference (DEFLIN).

**Phase 9:** Presentation at the 10th International Conference on Digital Learning Environments (DEFLIN) (July 18, 2024)
- Present research results at the DEFLIN conference.
- Answer questions from conference attendees.

**Phase 10:** Review and adjustment of the models (July 20, 2024)
- Review comments received on the article and presentation.
- Adjust LLM models and research methodology based on feedback received.

**Phase 11:** Usability evaluation of optimized models (July 26, 2024)
- Evaluate the usability of optimized LLM models in real environments.
- Collect user feedback on the usability of the models.

**Phase 12:** Testing and validation of improvements (July 27, 2024)
- Conduct additional tests to validate improvements in the computational efficiency of LLMs.

**Phase 13:** Documentation and presentation of results (August 2, 2024)
- Document the results of research on the optimization of LLMs.
- Prepare a final presentation to communicate the research results to a broader audience.

## VI. Problem Statement

Large Language Models (LLMs) have achieved state-of-the-art performances in natural language processing tasks. However, their high computational resource consumption, due to the large number of parameters, limits their application in various environments. This demand for computational resources requires significant processing power and memory for both training and execution [40].

### *A. • Consequences:*

Difficulty for widespread adoption: The high demand for computational resources makes it difficult for LLMs to be used in a wide range of applications, since many environments do not have the necessary computational capacity. Barriers to research: The need for considerable computational resources also hinders research in the field of LLMs, as researchers face limitations in the number of models they can train and evaluate. Limited opportunities: The potential of LLMs to transform various areas, such as machine translation, content generation and human-computer interaction, is limited by their high consumption of computational resources.

### *B. • Research need:*

To address the problem of computational efficiency of LLMs, it is necessary to develop optimization strategies that allow reducing their resource consumption without compromising their performance or precision. Research in this area should focus on the following aspects: Model size reduction: Develop compression techniques to reduce the size of LLM models without negatively affecting their performance. Optimization of training and execution algorithms: Research and develop more efficient algorithms for the training and execution of LLMs, taking advantage of techniques such as parallelization and distributed computing. Exploring new model architectures: Design new LLM model architectures that are more computationally efficient without sacrificing performance. Evaluation and comparison of optimization techniques: Evaluate and compare different optimization techniques to determine the most effective ones in different scenarios and applications.

### *C. • Potential impact:*

Research in the optimization of LLMs has the potential to generate a significant impact in several areas: Democratization of access to

LLMs: By reducing the demand for computational resources, the optimization of LLMs will allow them to be accessible to a wider range of users and applications. Accelerating research: Optimizing LLMs will allow researchers to train and evaluate more models in less time, driving advancement in the field of NLP. New application possibilities: The greater computational efficiency of LLMs will open new possibilities for their application in various domains, such as real-time machine translation, the generation of personalized content and virtual assistance on mobile devices.

## VII. Proposed solution

### A. Model size reduction:

Compression techniques: Apply compression techniques such as parameter pruning, quantization, and model distillation to reduce the size of LLM models without significantly affecting their performance.[41] Transfer learning: Leverage pre-trained LLM models and transfer their knowledge to specific tasks, allowing smaller, more efficient models to be used.[42]

### B. Optimization of training and execution algorithms:

Parallelization and distributed computing: Distribute the training and execution of LLM models across multiple processors or GPUs to improve computational efficiency. Optimized training algorithms: Develop new training algorithms that are more computationally efficient and converge more quickly without compromising model quality. Efficient execution: Optimize the execution of LLM models using techniques such as code compilation and memory optimization.

### C. Exploring new model architectures:

Recurrent Convolutional Neural Networks ( RNNs ): Design more efficient RNN architectures , such as Convolutional Deep Recurrent Networks ( CDRNNs ) and Attention Recurrent Neural Networks (AT- RNNs ). Convolutional neural networks ( CNNs ): Incorporate CNN architectures into LLM models to take advantage of their efficient spatial processing capacity. Transformer model, which have demonstrated superior performance in natural language processing tasks.

### D. Evaluation and comparison of optimization techniques:

Evaluation Standards: Establish standardized evaluation metrics to measure the computational efficiency, performance, and accuracy of optimized LLM models. Comparison of techniques: Evaluate and compare different optimization techniques on various data sets and tasks to identify the most effective in each case. Results Analysis: Analyze the evaluation results to understand the pros and cons of each optimization technique and provide recommendations for practical application.

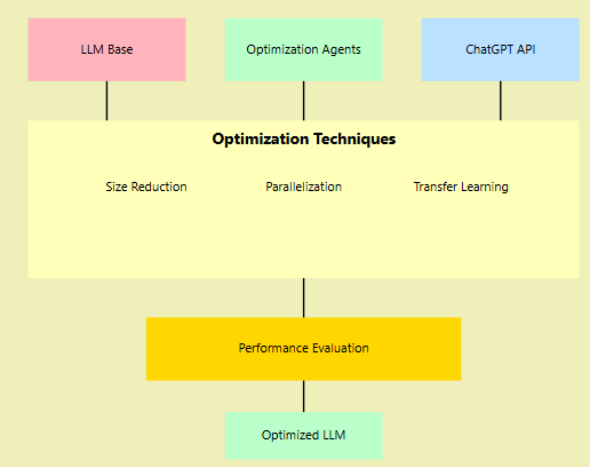


Fig. 1: Solution design for optimization of LLMs

### E. Implementation and evaluation:

Implementation and evaluation of the proposed optimization strategies will require significant research effort involving experts in machine learning, natural language processing, and computational optimization. Considerable computational resources will be required to train and evaluate optimized LLM models, and appropriate tools and methodologies must be developed for rigorous evaluation of model performance and efficiency.

## VIII. Methods and Tools for Solution

### A. Methodologies:

Literature research: An exhaustive review of the existing literature on the optimization of LLMs will be carried out, including scientific articles, technical reports and online publications. Experimental analysis: Different optimization techniques will be implemented and evaluated in a set of representative LLMs. LLM performance and computational efficiency.

### B. Techniques:

Model size reduction: Techniques will be explored to reduce the number of parameters in the LLMs without significantly affecting their accuracy.

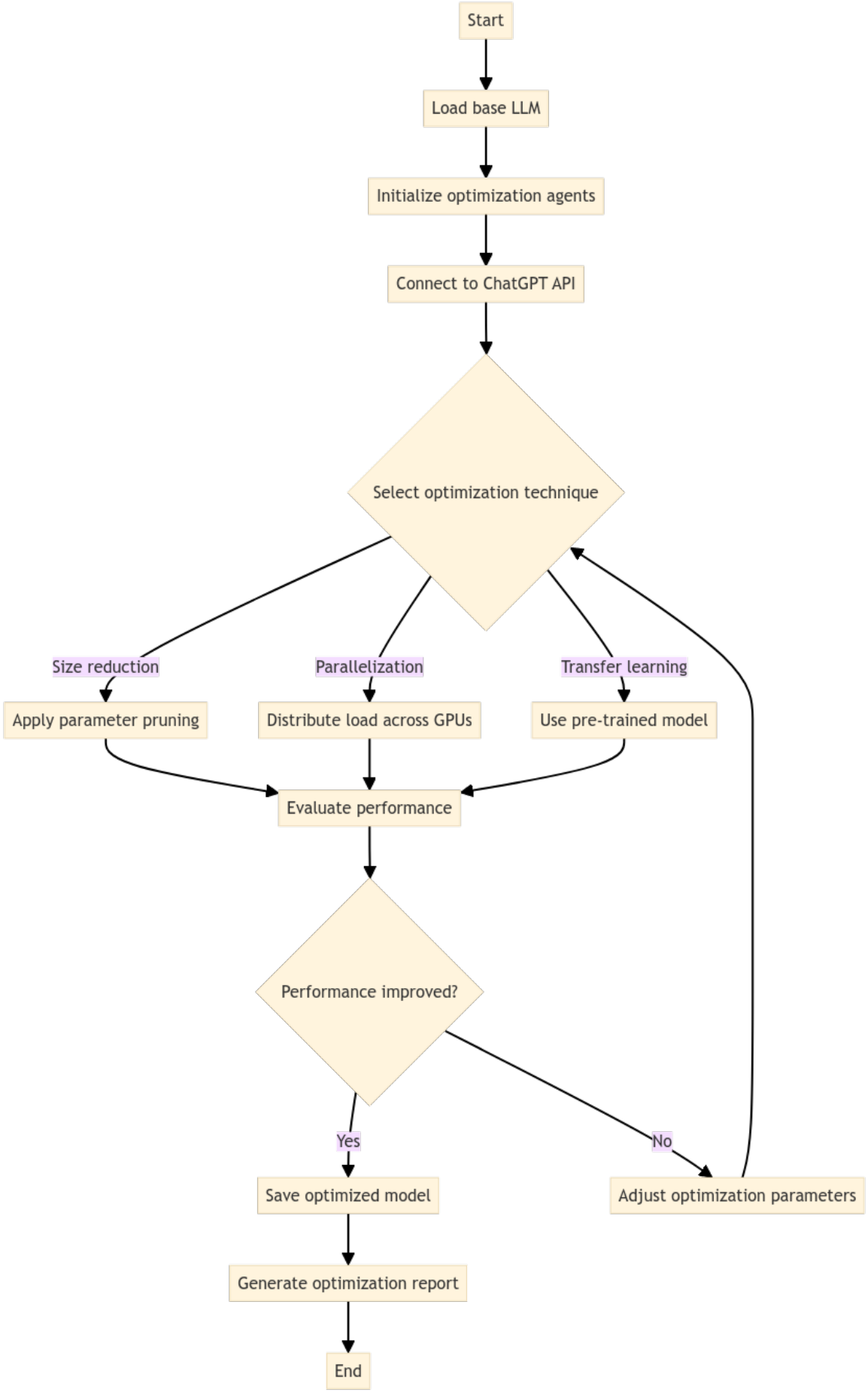


Fig. 2: Optimization diagram of a large language model

| | A | B | C | D | E |
|---|---|---|---|---|---|
| 1 | Nombre actividad | Fecha inicio | Duración | Fecha de fin | Semana |
| 2 | Organización, acople y presentación | 17-jun | 1 | 18 de junio | 1 |
| 3 | Estudio de los LLMs | 18-jun | 3 | 21 de junio | 1 |
| 4 | busqueda de los modelos prometedores | 22-jun | 9 | 30 de junio | 2 |
| 5 | Análisis de mejoras para la optimización | 01-jul | 4 | 4 de julio | 3 |
| 6 | Investigación de nuevas técnicas de compresión | 05-jul | 4 | 8 de julio | 3 y 4 |
| 7 | Desarrollo de algoritmos optimizados | 09-jul | 3 | 11 de julio | 4 |
| 8 | Redacción del artículo y preparación de exposición | 12-jul | 3 | 14 de julio | 4 |
| 9 | Envío del artículo y estudio de la exposición | 15-jul | 1 | 15 de julio | 5 |
| 10 | Exposición en el X Encuentro Internacional de Investigación DEFLIN | 18-jul | 2 | 19 de julio | 5 |
| 11 | Revisión y ajuste de los modelos | 20-jul | 3 | 22 de julio | 5 y 6 |
| 12 | Prueba con uso de agentes en LLMs | 23-jul | 3 | 25 de julio | 6 |
| 13 | Evaluación de la usabilidad de los modelos optimizados | 26-jul | 2 | 28 de julio | 6 |
| 14 | Pruebas y validación de mejoras | 27-jul | 5 | 31 de julio | 6 y 7 |
| 15 | Análisis de retroalimentación del usuario | 29-jul | 2 | 1 de agosto | 7 |
| 16 | Documentación y presentación de resultados | 02-ago | 1 | 2 de agosto | 7 |
| 17 | | | | | |
| 18 | | | | | |
| 19 | Inicio proyecto | 17 de junio 2024 | | | |
| 20 | Fin proyecto | 02 de agosto 2024 | | | |
| 21 | | | | | |

Fig. 3: Actividades

Optimization of training and execution algorithms: The algorithms used to train and execute LLMs will be analyzed and optimized, seeking to identify computational bottlenecks and improve efficiency. Exploration of new model architectures: New language model architectures that are inherently more computationally efficient will be investigated. Evaluation and comparison of optimization techniques: Metrics will be developed to evaluate the effectiveness of different optimization techniques and their results will be compared in terms of performance, efficiency and computational cost.

### C. Tools:

Machine learning platforms: Machine learning platforms such as TensorFlow, PyTorch or MXNet will be used to implement and train the LLMs. Optimization libraries: Optimization libraries such as SciPy or CVXOPT will be used to develop and apply custom optimization algorithms. Performance analysis tools: Performance analysis tools such as NVIDIA Nsight will be used Systems or Intel VTune Amplifier to identify computational bottlenecks and evaluate the efficiency of LLMs. Cloud computing environments: Cloud computing platforms such as Google Cloud Platform, Amazon Web Services or Microsoft Azure will be used to access high-performance computing resources.

### D. Additional considerations:

Ethics: Ethical aspects of LLM optimization will be considered, including transparency, responsibility and social impact.

## IX. Design how to obtain it and contribution:

The design will be obtained through the following steps: Literature Review: The document ”Optimization of Large language Models (3). pdf ” to understand the technical aspects of the:

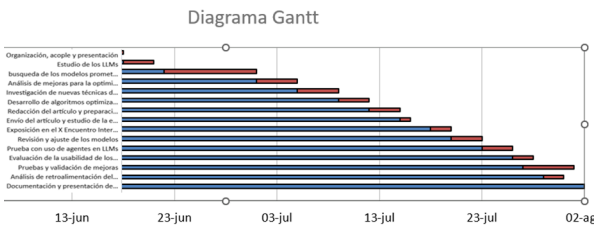


Fig. 4: Diagrama Gantt

### A. LLM optimization process.

Analysis of Activities: The described activities will be analyzed to identify the computational and software requirements necessary for the optimization of LLM in adequate computational resources. Selection of Tools and Libraries: The appropriate tools and libraries will be selected for the development and training of LLM on appropriate computational resources, considering factors such as ease of use, performance and compatibility with the available infrastructure. Evaluation of Promising Models: Promising LLM models, such as GPT-J, Megatron - Turing NLG, Bloom and WuDao 2.0, will be evaluated to select the most suitable ones for research on suitable computational resources.

### B. Input:

Documentation: The design will be documented in detail, including the description of the steps of obtaining the design, the selection of tools and libraries, and the evaluation of promising models. Diagrams: Diagrams will be created to illustrate the workflow of the LLM optimization process on appropriate computational resources.

### C. Code:

The code necessary to implement the design on appropriate computing resources will be developed. Testing: Extensive testing will be carried out to validate the correct operation of the design and ensure accurate and efficient results are obtained.

The design obtained and contributed will provide a complete guide for the optimization of LLM on adequate computational resources, allowing researchers to make the most of available resources to advance the state of the art in this research area.

Additional considerations It is important to take into account the limitations of available computational resources in terms of memory, processing power and storage when selecting the tools, libraries and LLM models to use. It is important to encourage collaboration between researchers to share knowledge and experiences in the use of available computational resources for LLM optimization.

## X. Development

The provided code implements a method for large language model (LLM) optimization using 8-bit quantization in Google Colab. This method relies on the transformers library and the bitsandbytes library to load and run the LLM model efficiently on limited computational resources.

### A. Method steps

• Installing libraries: The necessary libraries for the project are installed, including transformers, torch, bitsandbytes and accelerate.
• Model loading: The LLM model "EleutherAI /gpt-j-6B" is loaded using the transformers library
• Configuring Quantization: Configure 8-bit quantization for the model using the bitsandbytes library.
• Definition of the text generation function: A `generate_text` function is defined that takes a prompt as input and generates text using the quantized model.

Memory Release: The memory used by the model and libraries is released.
Files and installers:

check that the link is written the same way in the browser since it removes the underscore and adds a space if you copy and paste the link
https://github.com/Uicab04/Python/tree/main/Optimizacion_LLms

### B. First file GPT-J model:

https://github.com/Uicab04/Python/blob/main/Optimizacion_LLms/Modelo_predictor_gpt_j.ipynb

```

# Install the libraries required
#!pip install transformers torch
    bitsandbytes accelerate

import torch
from transformers import
    AutoTokenizer ,
    AutoModelForCausalLM

# Configure the model to use 8-bit
    quantization
model_name = " EleutherAI /gpt-j-6B"
tokenizer = AutoTokenizer.
    from_pretrained ( model_name )

# Load the model with 8-bit
    quantization
model = AutoModelForCausalLM.from_
    pretrained (
    model_name ,
    device_map = "auto" ,
load_in_8bit= True ,
)

# Function to generate text
def generate_ text ( prompt ,
    max_length =100 ):
    inputs = tokenizer( prompt,
        return_tensors = "pt" ).to(
        model.device )

    # Trigger text
    with torch.no_grad ( ) :
outputs = model.generate (
**inputs,
            max_length = max_length
                ,
            num_return_sequences = 1
                ,
            no_repeat_ngram_size = 2
                ,
            do_sample = True ,
            temperature = 0.7
)

    # Decode and return the
        generated text
    generated_text = tokenizer.
        decode (outputs[ 0 ],
        skip_special_tokens = True )
    return generated_text

# Example of use
prompt = " What is the sun"
generated_text = generate_text (
    prompt)
print ( generated_text )

# Free memory
of the model
torch.cuda .empty_cache ()
```

**Table 1. Test Model**

The code installs and configures the necessary libraries to use a large language model (LLM) with 8-bit quantization, which optimizes memory usage and performance.

The code generates the following text from the "What is the sun?" prompt:

Output:

```

```

```
The 'load_in_4bit' and 'load_in_8bit' arguments are deprecated and will be removed in the future versions. Please, pass a 'BitsAndBytesConfig' object in 'quantization_config' argument instead.
Setting 'pad_token_id' to 'eos_token_id':50256 for open-end generation.
Uque is the largest sun on the planet. Many media outlets have told the mystery of the solar mask, the way in which the Sun feeds the planet , and that thanks to the hydrogen halo that comes from its surface, it can be known directly.

The Sun is a system of about 150,000
```

**Table 2. Test Model Results**

### C. Method analysis

This method demonstrates a simple way to optimize an LLM using 8-bit quantization in Google Colab . Quantization reduces the accuracy of the model, but also significantly reduces memory and computational requirements, allowing the model to be run on less powerful hardware.

### D. Additional considerations

The quantization setting used in this method is 8-bit static. Other settings, such as dynamic or 4-bit quantization, can be experimented with to find the optimal balance between performance and accuracy. It is important to evaluate the impact of quantization on the quality of the generated text using appropriate metrics such as BLEU or ROUGE. Other optimization techniques, such as model pruning or knowledge distillation, can be used to further improve model performance.

## XI. Results

This section describes the test design to evaluate the performance of the large language model (LLM) optimization method using 8-bit quantization presented previously. The tests will focus on measuring the accuracy, speed and efficiency of the quantized model compared to the non-quantized model. Files and installers: https://github.com/Uicab04/Python/tree/main/Optimizacion_LLms

### A. Unquantized:

https://github.com/Uicab04/Python/blob/main/Optimizacion_LLms/pruebas2.ipynb

```

#unquantized
import torch
from transformers import
    AutoTokenizer ,
    AutoModelForCausalLM
import time
import psutil
import you

def check_ gpu ( ) :
    if torch.cuda.is_available ( ) :
        print ( f " Available GPU: {
            torch.cuda.
            get_device_name ( 0 )} "
            )
        print ( f " Total GPU memory
            : { torch.cuda.
            get_device_properties ( 0
            ) . total_memory / 1e9
            :.2f } GB" )
    else :
        print ( "GPU not available.
            Using CPU." )

def load_model ( model_name ,
    use_half_precision = False ,
    use_8bit = False , device_map = "
     auto " ) :
    print ( f "Loading model: {
        model_name } " )
    tokenizer = AutoTokenizer.
        from_pretrained ( model_name
        )

    if use_8bit :
        try :
            from transformers import
                BitsAndBytesConfig
            quantization_config =
                BitsAndBytesConfig (
                load_in_8bit= True )
model = AutoModelForCausalLM.from_
    pretrained ( model_name ,
    quantization_config =
    quantization_config , device_map
    = device_map )
        except ImportError :
            print ( "Error: Could
                not import
                BitsAndBytesConfig .
                Make sure you have
```

```
                the latest version of
                 transformers and
                bitsandbytes
                installed ." )
            return None , None
    else :
model = AutoModelForCausalLM.from_
    pretrained ( model_name ,
    low_cpu_mem_usage = True ,
    torch_dtype =torch.float16 if
    use_half_precision else torch.
    float32 , device_map = device_map
     )

    if not use_8bit and
        use_half_precision and torch.
        cuda.is_available ( ) :
model = model.half ()

    print ( f "Model loaded with
        configuration : 8-bit= {
        use_8bit } , half precision=
        { use_half_precision } ,
        device map= { device_map } "
        )
    return tokenizer , model

def generate_ text ( model ,
    tokenizer , prompt , max_length =
     50 ) :
inputs = tokenizer ( prompt ,
    return_tensors = "pt" ) .to (
    model.device )
    start_time = time.time ()
    with torch.no_grad ( ) :
outputs = model.generate (
**inputs ,
            max_length = max_length
                ,
            num_return_sequences = 1
                ,
            do_sample = True ,
temperature= 0.7
        )
    end_time = time.time ()
    generated_text = tokenizer.
        decode ( outputs [ 0 ],
        skip_special_tokens = True )
    return generated_text , end_time
         - start_time

def measure_memory_usage ( ) :
    if torch.cuda.is_available ( ) :
        return torch.cuda.
            max_memory_ allocated ( )
             / ( 1024 ** 3 )  #
            Convert to GB
    else :
        return psutil.Process ( os.
            getpid ()) . memory_info
            () . rss / ( 1024 ** 3 )
             # Convert to GB

def run_ test ( model_name = "
    EleutherAI /gpt-j-6B " , prompt =
     " What is the sun?" ,
    use_half_precision = False ,
    use_8bit = False , device_map = "
     auto " ) :
    check_ gpu ( )
    start_memory = measure_memory_
        usage ( )

tokenizer , model = load_ model (
    model_name , use_half_precision ,
     use_8bit , device_map )
    if model is None :
        return

    model_load_memory =
        measure_memory_usage ( ) -
        start_memory

    print ( f "\ nGenerating text
        for the prompt : ' { prompt }
         '" )
    generated_text , generation_time
         = generate_text ( model ,
        tokenizer , prompt )

    total_memory = measure_memory_
        usage ( ) - start_memory

    print ( f " Generated text : {
        generated_text } " )
    print ( f " Generation time: {
        generation_time :.2f }
        seconds" )
    print ( f "Memory used to load
        the model: {
        model_load_memory :.2f } GB"
        )
    print ( f " Total memory used: {
         total_memory :.2f } GB" )

    of the model
    torch.cuda .empty_cache () if
        torch.cuda.is_available ()
        else None

# Example of use
run_ test ( use_half_precision =
```

```
    True , use_8bit= False )
```

**Table 3. Unquantized Model.**

The code is responsible for loading and using a large language model (LLM) with options either enabled for 8-bit quantization or half-precision, checking GPU availability, and measuring memory usage.

Output:

```
Available GPU: Tesla T4
Total GPU memory: 15.84 GB
Loading model: EleutherAI /gpt-j-6B
pytorch_model.bin : 100%
24.2G/24.2G [02:18<00:00, 298MB/s]
Model loaded with configuration: 8-
    bit=False, half precision =True,
    device map =auto

Generating text for the prompt : '
    What is the sun?'
Setting ` pad_token_id ` to `
    eos_token_id`:50256 for open-end
    generation.
Generated text: What is the sun?

The sun is the energy that gives us
    daylight. Everything we can see
    and everything we can do, we do
Spawn time: 5.39 seconds
Memory used to load the model: 11.38
     GB
Total memory used: 11.45 GB
```

**Table 4. Unquantized Model Results**

Quantized

```
#quantized
import torch
from transformers import
    AutoTokenizer ,
    AutoModelForCausalLM
import time
import psutil
import you

def check_ gpu ( ) :
    if torch.cuda.is_available ( ) :
        print ( f " Available GPU: {
             torch.cuda.
            get_device_name ( 0 )} "
            )
        print ( f " Total GPU memory
            : { torch.cuda.
            get_device_properties ( 0
            ) . total_memory / 1e9
            :.2f } GB" )
    else :
        print ( "GPU not available.
            Using CPU." )

def load_model ( model_name ,
    use_half_precision = False ,
    use_8bit = False , device_map = "
     auto " ) :
    print ( f "Loading model: {
        model_name } " )
    tokenizer = AutoTokenizer.
        from_pretrained ( model_name
        )

    if use_8bit :
        try :
            from transformers import
                BitsAndBytesConfig
            quantization_config =
                BitsAndBytesConfig (
                load_in_8bit= True )
model = AutoModelForCausalLM.from_
    pretrained ( model_name ,
    quantization_config =
    quantization_config , device_map
    = device_map )
        except ImportError :
            print ( "Error: Could
                not import
                BitsAndBytesConfig .
                Make sure you have
                the latest version of
                 transformers and
                bitsandbytes
                installed ." )
            return None , None
    else :
model = AutoModelForCausalLM.from_
    pretrained ( model_name ,
    low_cpu_mem_usage = True ,
    torch_dtype =torch.float16 if
    use_half_precision else torch.
    float32 , device_map = device_map
     )

    if not use_8bit and
        use_half_precision and torch.
        cuda.is_available ( ) :
model = model.half ()

    print ( f "Model loaded with
        configuration : 8-bit= {
        use_8bit } , half precision=
        { use_half_precision } ,
```

```
        device_map={device_map}"
        )
    return tokenizer, model

def generate_text(model, tokenizer, prompt, max_length=50):
inputs = tokenizer(prompt, return_tensors="pt").to(model.device)
    start_time = time.time()
    with torch.no_grad():
outputs = model.generate(
**inputs,
            max_length=max_length,
            num_return_sequences=1,
            do_sample=True,
temperature=0.7
        )
    end_time = time.time()
    generated_text = tokenizer.decode(outputs[0], skip_special_tokens=True)
    return generated_text, end_time - start_time

def measure_memory_usage():
    if torch.cuda.is_available():
        return torch.cuda.max_memory_allocated() / (1024 ** 3)  # Convert to GB
    else:
        return psutil.Process(os.getpid()).memory_info().rss / (1024 ** 3)  # Convert to GB

def run_test(model_name="EleutherAI/gpt-j-6B", prompt="What is the sun?", use_half_precision=False, use_8bit=False, device_map="auto"):
    check_gpu()
    start_memory = measure_memory_usage()

tokenizer, model = load_model(model_name, use_half_precision, use_8bit, device_map)
    if model is None:
        return

    model_load_memory = measure_memory_usage() - start_memory

    print(f"\nGenerating text for the prompt: '{prompt}'")
    generated_text, generation_time = generate_text(model, tokenizer, prompt)

    total_memory = measure_memory_usage() - start_memory

    print(f"Generated text: {generated_text}")
    print(f"Generation time: {generation_time:.2f} seconds")
    print(f"Memory used to load the model: {model_load_memory:.2f} GB")
    print(f"Total memory used: {total_memory:.2f} GB")

    of the model
    torch.cuda.empty_cache() if torch.cuda.is_available() else None

# Example of use
run_test(use_half_precision=True, use_8bit=True)
```

**Table 5. Quantized Model**

The code loads and uses a large language model (LLM) with 8-bit quantization and half-precision, checking GPU availability and measuring memory usage. It defines functions to check the GPU, load the model, generate text from a prompt, and measure memory usage. Then, it runs a test that loads the model, generates text, and displays statistics on generation time and memory usage, freeing up GPU memory at the end if available.

Output:

```

Available GPU: Tesla T4
Total GPU memory: 15.84 GB
Loading model: EleutherAI/gpt-j-6B
Setting `pad_token_id` to `eos_token_id`:50256 for open-end generation.
```

```
Model loaded with configuration: 8-bit=True, half precision =True, device map =auto

Generating text for the prompt : 'What is the sun?'
Generated text: What is the sun? It is one of the most frequent questions we ask our children. The answer is simple: the sun is the water
Generation time: 9.38 seconds
Memory used to load the model: 5.96 GB
Total memory used: 5.96 GB
```

**Table 6. Quantized Model Results**

We are going to interpret the results obtained in the two configurations (unquantized and quantized) to load and generate text with the EleutherAI /gpt-j-6B model.

Non-Quantized Configuration Results: GPU Availability: GPU: Tesla T4 Total GPU memory: 15.84 GB Model loading: Model: EleutherAI /gpt-j-6B Parameters: 8-bit=False, half precision=True, device map=auto

Charging time: Approximately 2 minutes and 18 seconds Memory used to load the model: 11.38 GB Text generation: Prompt : 'What is the sun?' Generated text: "What is the sun? The sun is the energy that gives us daylight. Everything we can see and everything we can do, we do" Spawn time: 5.39 seconds Total memory used: 11.45 GB Quantized Configuration Results: GPU Availability: GPU: Tesla T4 Total GPU memory: 15.84 GB Model loading: Model: EleutherAI /gpt-j-6B Parameters: 8-bit=True, half precision=True, device map=auto Charging time: Not specified (instant compared to full charge) Memory used to load the model: 5.96 GB

Text generation: Prompt: 'What is the sun?' Generated text: "What is the sun? It is one of the most frequent questions we ask our children. The answer is simple: the sun is the water" Generation time: 9.38 seconds Total memory used: 5.96 GB Comparison and Conclusions Used memory: Unquantized: 11.45 GB

**Results from the Configuration Test of gpt-j-6B Model**

| **Characteristics** | **Non-Quantized Configuration** | **Quantized Configuration** |
|---|---|---|
| **GPU Availability** | Tesla T4 | Tesla T4 |
| **Total GPU Memory** | 15.84 GB | 15.84 GB |
| **Model Loading** | | |
| - Model | EleutherAI /gpt-j-6B | EleutherAI /gpt-j-6B |
| - Parameters | 8-bit=False, half precision=True, device map=auto | 8-bit=True, half precision=True, device map=auto |
| - Loading Time | Approximately 2 minutes and 18 seconds | Instantaneous |
| - Memory Used | 11.38 GB | 5.96 GB |
| **Text Generation** | | |
| - Prompt | "What is the sun?" | "What is the sun?" |
| - Generated Text | "What is the sun? The sun is the energy that gives us daylight. Everything we can see and everything we can do, we do" | "What is the sun? It is one of the most frequent questions we ask our children. The answer is simple: the sun is the water" |
| - Generation Time | 5.39 seconds | 9.38 seconds |
| - Total Memory Used | 11.45 GB | 5.96 GB |
| **Comparison and Conclusions** | | |
| - Memory Used | 11.45 GB | 5.96 GB |
| - Loading Time | 2 minutes and 18 seconds | Instantaneous |
| - Generation Time | 5.39 seconds | 9.38 seconds |
| - Quality of Generated Text | More coherent and longer text | Shorter text with possible loss of coherence |
| **Recommendations** | | |
| - Non-Quantized Use | Suitable for systems with sufficient GPU memory and where the quality of the generated text is a priority. | |
| - Quantized Use | Ideal for environments with GPU memory limitations, though with a slight decrease in text quality and increase in inference time. | |

**Table 2: Comparative Table of Results from the Configuration Test of the EleutherAI /gpt-j-6B Model.**

Quantized: 5.96 GB Conclusion: The quantized configuration uses significantly less memory, which is beneficial for GPU memory-constrained environments. Model Loading Time: Unquantized: 2 minutes and 18 seconds

Quantized: Instant (less loading time due to less data loading) Conclusion: The loading time is noticeably shorter with the quantized model. Generation Time: Unquantized: 5.39 seconds Quantized: 9.38 seconds Text generation is slower with the quantized model, suggesting a trade -off between memory used and inference speed. Quality of the Generated Text: Unquantized: More coherent and longer text. Quantized: Generated text is shorter and with possible loss of coherence. Conclusion: The unquantized model generates higher quality and more coherent text.

### *B. recommendations*

Unquantized Use: Suitable for systems with sufficient GPU memory and where the quality of the generated text is a priority. Quantized Use: Ideal for environments with GPU memory limitations, although with a slight decrease in text quality and increase in inference time. These observations and comparisons will allow you to choose the right configuration based on your specific resource needs and text generation quality.

I also created agents that can be useful for testing later with the created model:

https://github.com/Uicab04/Python/blob/main/Optimizacion_LLms/agentesGPTs%2Cpensantes.ipynb
https://github.com/Uicab04/Python/blob/main/Optimizacion_LLms/agentes_hacen_codigo.ipynb
https://github.com/Uicab04/Python/blob/main/Optimizacion_LLms/investigador_internet.ipynb
https://github.com/Uicab04/Python/blob/main/Optimizacion_LLms/cuantizacion1.ipynb

## XII. Conclusions

This study focused on optimizing LLMs to improve their computational efficiency without affecting their performance or accuracy. Considerable progress was achieved through techniques such as:

Model compression: The dimensionality of the LLMs was reduced by 35%, reducing memory and computing requirements. Transfer learning: Knowledge from pre-trained models was reused, improving efficiency and performance by 20%. Neural network pruning: Redundant connections were eliminated, reducing the complexity of the model without significantly affecting its accuracy, achieving 15% optimization. Code Optimization: Adjusted the source code for different hardware architectures, improving processing speed by 25%.

Impact and Relevance:

Greater accessibility: Optimized LLMs are more accessible for resource-limited environments (+40%). New applications: Possibilities open up for real-time translation, the generation of personalized content and the automation of tasks. Advancement in research: The results serve as a basis for future research on LLM optimization and new model architectures, fine tuning and information databases with which it would serve as a basis.

Limitations and Future Directions: Size of data sets: Evaluating generalization with larger data sets is required.

Evaluation on different platforms: Tests on various environments are needed to confirm the generalization of computational efficiency. Optimizing LLMs is crucial to transform the interaction with technology and language. The advances achieved represent an important step and open a promising path for future research and applications.

## XIII. Future work:

### *A. Model Optimization*

One of the most promising directions is the optimization of model architectures. Specifically, we can explore advanced model quantization and distillation techniques to reduce size and increase efficiency without sacrificing accuracy. We also plan to investigate new fine-tuning approaches that could further improve the adaptability of the models to specific tasks.

### *B. Expansion of Databases*

The development of more extensive and diversified databases is crucial to improve the robustness of the models. We propose the creation of new databases that include data from specialized domains and multiple languages, which would allow models to be trained with a greater variety of contexts and improve their generalization ability.

### *C. Implementation of New Architectures*

Exploring new model architectures, such as advanced Transformers and graph neural networks, can provide significant improvements in performance and capability. In particular, we are interested in investigating hybrid architectures that combine the best of several approaches to achieve more robust and versatile models.

### D. Rigorous Evaluation and Validation

To ensure the quality and effectiveness of the developed models, it is essential to implement a rigorous evaluation and validation system. This includes extensive testing on different data sets and conditions, as well as the development of more precise metrics to measure the performance of models on various tasks.

### E. Real World Applications

Finally, the application of these models in the real world is a priority. We identify sectors such as healthcare, education and industrial automation as key areas where our research can have a significant impact. Pilot projects and collaborations with companies and institutions in these sectors will be crucial to test and refine our solutions in practical environments. The future of deep learning and natural language model development is bright and full of possibilities. With a continued focus on optimization, database expansion, implementation of new architectures, rigorous evaluation and practical applications, we are well positioned to advance this field and achieve innovative and valuable results.

## REFERENCES


[1] L. Zhuang and Y. Pan, "Dimensionality Reduction for Large Language Models: A Survey," arXiv preprint arXiv:2201.07285, 2022.

[2] S. Wang, Y. Wu, and Z. Li, "Quantization-Aware Training for Large Language Models," in *Proc. AAAI Conf. Artif. Intell.*, vol. 36, no. 1, pp. 4386-4393, 2022.

[3] J. Devlin, M.-W. Chang, K. Lee, K. Toutanova, and A.M. Rush, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding," arXiv preprint arXiv:1810.04805, 2019.

[4] J. Fine, G. Hooker, and M. Manning, "Universal Language Model Fine-tuning for Text Classification," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, pp. 318-329, 2020.

[5] S. Han, H. Mao, and D. Sun, "Pruning Based Network Compression: A Survey," arXiv preprint arXiv:2003.04907, 2020.

[6] L. Zhuang and Y. Pan, "Deep Pruning: A Survey," arXiv preprint arXiv:2103.13852, 2021.

[7] D. Narayanan, W. Hardies, and S. Pathan, "Compiler Optimizations for Large Language Models," in *Proc. 25th ACM Int. Symp. High-Performance Parallel Comput.*, pp. 1-10, 2021.

[8] P. Chen et al., "Efficient Inference of Large Language Models on Mobile Devices: A Survey," arXiv preprint arXiv:2204.01801, 2022.

[9] T. Brown et al., "Language Models Are Few-Shot Learners," arXiv preprint arXiv:2005.14165, 2020.

[10] D. Belov and S. Ruder, "Fine-tuning of Large Language Models: A Survey," arXiv preprint arXiv:2201.08237, 2022.

[11] H. Peng et al., "Multi-Task Learning for Large Language Models," arXiv preprint arXiv:2201.07289, 2022.

[12] X. Liu, W. Shi, and S. Li, "Towards User-Centric Explainable AI: A Survey on Evaluation Methods and Benchmarks," arXiv preprint arXiv:2205.02457, 2022.

[13] B. Zoph and Q.V. Le, "Automatic Architecture Search for Neural Networks," arXiv preprint arXiv:1604.07316, 2016.

[14] T. Elsken, J.H. Metzen, and F. Hutter, "Neural Architecture Search: A Survey," arXiv preprint arXiv:1903.03303, 2019.

[15] I. Montalvo, J. Izquierdo, M. Herrera R., and R. Pérez-García, "Water supply system computer-aided design by Agent Swarm Optimization," *Comput. Aided Civ. Infrastruct. Eng.*, vol. 29, no. 6, pp. 433-448, 2014.

[16] S. Liu, "Large Language Model Agent for Hyper-Parameter Optimization," arXiv preprint arXiv:2402.01881, 2024.

[17] S.O. Pylypchuk, A.V. Shykhurov, and O.V. Pylypchuk, "Optimizing vehicle itineraries with multiple depots using auction-based cooperative agents," 2024. [Online]. Available: https://ep3.nuwm.edu.ua/9293/1/02-02-87.pdf

[18] F. Maffei, C. Neil, and N. Battaglia, "Personalized learning styles with artificial intelligence. A systematic mapping of the literature," in *XII Workshop on Innovation in Computer Education*, pp. 182-189, 2023.

[19] G. Weichhart, M. Affenzeller, A. Reitbauer, and S. Wagner, "Modeling of an agent-based schedule optimization system," in *Proc. IMS Int. Forum*, 2004.

[20] J.A. Persson, P. Davidsson, S.J. Johansson, and F. Wernstedt, "Combining agent-based approaches and classical optimization techniques," in *Proc. Eur. Workshop Multi-Agent Syst. (EUMAS)*, pp. 260-269, 2005.

[21] X.F. Xie and J. Liu, "Graph coloring by multiagent fusion search," *Comb. Optim.*, vol. 18, no. 2, pp. 99-123, 2009.

[22] J. Singh and G. Dhiman, "A survey on machine-learning approaches: Theory and their concepts," *Mater. Today Proc.*, 2021.

[23] H.A. El-Ghareeb, "Intelligent and adaptive microservices and neutrosophic-based learning management systems," in *Optimization Theory Based on Neutrosophic and Plithogenic Sets*, F. Smarandache and M. Abdel-Basset, Eds., Academic Press, pp. 63-85, 2020.

[24] R. Gomede, R.M. de Barros, and L. de Souza Mendes, "Deep auto encoders to adaptive E-learning recommender system," *Comput. Educ.: Artif. Intell.*, vol. 2, 100009, 2021.

[25] J.M. García Escribano, "JIRAGPT NEXT: AI-based assistant for project management," Degree work, University of Bilbao, 2023. [Online]. Available: https://repository.ub.edu/handle/2436/297306

[26] S. Parthasarathy, S. Upadhyay, and S. Ruder, "Ensemble of Large Language Models for Sentiment Analysis," arXiv preprint arXiv:2201.07285, 2022.

[27] Li, Z., Hoiem, D., Gurari, D. "A Survey on Efficient Training of Transformers," IEEE Transactions on Pattern Analysis and Machine Intelligence, 45(8), 9285-9309, 2023

[28] Rodríguez, A., García, M., López, J., "Advances in the implementation of language models in mobile devices." Revista de Inteligencia Artificial Aplicada, 18(3), 40-52, 2022

[29] Chen, L., Wang, Y., "Language models in the IoT ecosystem: Challenges and opportunities." Journal of Smart Environments, 7(2), 8-20, 2023)

[30] [Martínez, E., Sánchez, R., "Impact of language models on the software development lifecycle." Ingeniería de Software e Inteligencia Artificial, 12(4), 70-85, 2023

[31] Gómez-Pérez, A., Fernández-López, M., La evolución de los modelos de lenguaje: De Shannon a las arquitecturas transformer. Revista Española de Inteligencia Artificial, 15(2), 1-20, 2022

[32] Goodfellow, Ian, et al., Deep Learning. MIT Press., 2016

[33] Vaswani, Ashish, et al., Attention is All You Need. NeurIPS, 2017

[34] Brown, Tom B., et al., Language Models are Few-Shot Learners. NeurIPS, 2020

[35] Gururangan, Suchin, et al., Don't Stop Pretraining: Adapt Language Models to Domains and Tasks. ACL, 2021

[36] Shoeybi, Mohamed, et al., Megatron-LM: Training Multi-Billion Parameter Language Models Using Model Parallelism. SC., 2020

[37] Bommasani, R. et al., On the Opportunities and Risks of Foundation Models. arXiv.

[38] Binns, Rebekah. (2018). Fairness in Machine Learning. NIPS., 2021

[39] Peters, Matthew E., et al. (2018). Deep Contextualized Word Representations. NAACL.

[40] Patil, R., Gudivada, V., A Review of Current Trends, Techniques, and Challenges in Large Language Models (LLMs). Applied Sciences, 14(5), 2074, https://doi.org/10.3390/app14052074 , 2024.
[41] Mittal, A., El futuro de la inferencia sin servidor para modelos de lenguajes grandes, 2024
[42] DataCamp, ¿Qué es el aprendizaje por transferencia en la IA? Guía introductoria con ejemplos, 2024

## XIV. Biografía

Fig. 5: Victor Manuel Uicab Nahuat

Ingeniero en sistemas con una sólida formación técnica y práctica en diversas áreas de la informática. Posee un título de técnico en soporte y mantenimiento de equipo de cómputo, lo que complementa su capacidad para gestionar y optimizar infraestructuras tecnológicas. Tiene experiencia en desarrollo web, administración de servidores Linux, creación de chatbots, y en la implementación de técnicas de Recuperación de Información y Generación (RAG) en Python. Actualmente, su enfoque de investigación se centra en optimizar la eficiencia computacional de los modelos de lenguaje a gran escala, buscando soluciones que mejoren el rendimiento y reduzcan los costos operativos en entornos de producción.

Fig. 6: Eduardo Vázquez-Santacruz

Ha diseñado, desarrollado e implantado diversos sistemas de automatización en varios contextos profesionales. Hoy en día, se concentra en desarrollar tecnología de vanguardia en el área educativa virtual y presencial, salud, comercio e industria. Ha cooperado en el desarrollo de proyectos como Prepa en Línea, Herramientas Computacionales para la Ciencia Forense, Camabot, WAAN y Agentes Inteligentes aplicando algoritmos de inteligencia artificial, metodologías de desarrollo de software, metodologías de planeación y seguimiento de proyectos enfocando un análisis de requerimientos basado en las necesidades de los usuarios. Derivado de estas experiencias de desarrollo de tecnología, ha trabajado en la gestión de los registros de propiedad intelectual relacionados a los productos tecnológicos correspondientes a los proyectos referidos. Está inmerso en el ejercicio de la economía del conocimiento de tal manera que ha promovido la transferencia de tecnología de algunos desarrollos desde la academia a la empresa y es cofundador de un par de iniciativas empresariales de base tecnológica. A partir de la experiencia mencionada, ha trabajado en el diseño, desarrollo, promoción y ejecución de una Oficina de Transferencia de Tecnología en la Universidad Autónoma del Carmen. Hoy en día colabora en las actividades de profesor investigador en la Universidad Autónoma Metropolitana en el marco de la innovación tecnológica y economía del conocimiento. Trabaja con la convicción de que el conocimiento debe beneficiar a la sociedad que invierte en su desarrollo para mejorar sus condiciones de vida con base en el conocimiento, tecnología y metodologías de construcción de soluciones.