End-of-studies internship ST30
Optimization and fine tuning of Large Langage Models (LLM)

Student
DAVOUSE **Nathan**

School Representative
MAKBOUL **Salma**

Academics Programs
Industrial Engineering
Systems Optimization and Safety

Semester
Fall 2024

Abstract (150 words)

This internship took place at INRIA Lille, inside the BONUS research team. The aims of this internship were to explore the applications of optimization algorithms to the current hot topic of Neural networks: Large Language Models (LLM). In particular, I worked on Hyper-Parameter Optimization (HPO) applied to LLM fine-tuning. The work was split in three stages:

- Definition of the subject: explore the specific literature to understand stakes and how it works.
- Implementation of the algorithms: develop HPO algorithms and link them to LLM fine-tuning.
- Experiments: refine the implementation along the results, to obtain relevant outcomes.

This semester was a first immersion in the research and academic fields, allowing me to nourish my professional project.

Company : Inria Lille

Adress : 172 avenue de Bretagne, 59000 Lille

Tutor : El-Ghazali Talbi

Keywords
- Recherche appliquée
- Transport et Télécommunications
- Informatique
- Optimisation mathématique
- Logiciels - Recherche

UNIVERSITÉ DE TECHNOLOGIE TROYES

Intentionnaly left blank

Acknowledgements

Hello, here is some text without a meaning. This text should show what a printed text will look like at this place. If you read this text, you will get no information. Really? Is there no information? Is there a difference between this text and some nonsense like “Huardest gefburn”? Kjift – not at all! A blind text like this gives you information about the selected font, how the letters are written and an impression of the look. This text should contain all letters of the alphabet and it should be written in of the original language. There is no need for special content, but the length of words should match the language.

Chapter 1
Subject Definition

Scientific inquiry starts with observation. The more one can see, the more one can investigate.

Martin Chalfie

In a research internship, or indeed in any research activity, it's crucial to properly define the subject of the research, and to extract a precise problematic. That's why, during my first few weeks as an intern, and in this chapter of the report, I've endeavored to define this framework. To do so, I begin by recalling the subject as defined in my agreement:

NAS for LLM architectures is computationally prohibitive. In this work, we will investigate the use of efficient optimization algorithms (example: parallel fractal optimization) to reduce the latency of real-world commercial web-scale text prediction system. The goal of this work is to solve the NAS problem to find an architecture that when trained with data D and training algorithm A, produces a model that has similar accuracy but significantly reduced latency. The tasks composting this work are summarized below:

- Modeling and analysis of the NAS problem
- Solving of the problem using original and high-performance optimization algorithms
- Application to well known LLM such as GPT
- Application to logistics/transports problems

From there, to understand fully the problems, it's important to define what are LLM, and what are the stakes of this fields. Then, we will take a look about the application to manufacturing or logistics contexts. Alike we will look at the Auto-DNN fields, with focus to NAS and HPO problems, to locate the further work in a global litterature. With this, I can finally close this part with a precise search problematic, to drive my contribution.

1.1 Large Language Models (LLM)

LLM can be defined as Neural Networks using the Transformer bloc for Natural Language Processing (NLP) problems. For the next parts, it's important to understand the architecture of the LLM, after a brief reminder about DNN. Due to computation limitation, a lot of research contribution are about the Fine-tuning, defined in 1.1.4. To pursue to the next section, I will finish with a light review and taxonomy of LLM.

1.1.1 Deep Neural Networks (DNN)

Like many fields, DNN comes from biology-inspired design. In 1943, Warren Mcculloch and Walter Pitts introduced the idea of logical calculus and computation, based on neural network, in article [**mcculloch_logical_1943**]. The Artificial Neural Networks (ANN) aims to reproduce the cells of the brain to make a reasoning : the brain neuron is a node using a function to "activate" and the synapse is edge linking neurons to each others.

The figure 1.1 show the structure of an artifical neuron, including the trainable weights ω_n and the bias b. The output of the neuron can be write as : $\hat{y} = f(x, \omega) = \sigma(\sum_i^n x_i \omega_i + b)$, with x_n the input, and $\sigma(.)$ the activation function.

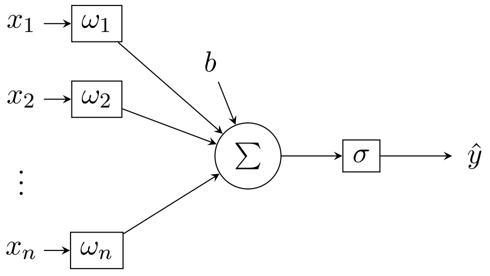

Figure 1.1: Illustration of an artifical neuron[**these_DNN**]

The activation function is the key of the ANN. The function need to make an output adapted to the use case (e.g. value between 0 and 1 for a probability), and differentiable to be able to use gradient-based optimization, the most efficient optimization method when possible. Approaches using Evolutionnary Algorithms (EA) to update weights were studied [**X**, **X**], and called *meta-learning*, but were deprecated by gradient-based methods.

With notation of figure 1.1, and ω being the matrix of all parameters, the training of the ANN can be expressed as equation 1.1. The loss $\mathcal{L}$ is the expression of the difference between the wanted output y and the predicted output $\hat{y}$. Many formulas can be used the compute the loss, such as cross-entropy [**X**] or Mean-Squared Error (MSE).

$$\omega \in \arg\min_{\omega \in \mathbb{R}^n} \mathcal{L}(\hat{y}, y) \tag{1.1}$$

The optimization of equation 1.1 is done using gradient descent, and especially Stochastic Gradient Descent (SGD). For each parameter, the gradient is expressed as 1.2. Based on this, the parameter is updated with $\omega_i \leftarrow \omega_i - \eta * \Delta_{\omega_i}$

$$\Delta_{\omega_i} = \Delta_{\omega_i}(x, y) = \frac{\partial \mathcal{L}(f(x, \omega), y)}{\partial \omega_i} \tag{1.2}$$

To accelerate the gradient descent, some optimization algorithm like Adaptive Moment Estimation (Adam) [1] use the momentum of the function, replacing $\Delta\omega_i$ with $m_t = \beta m_{t-1} + (1 - \beta)[\Delta\omega_i]$. In this formula, m_t is the moment of the function L at the instant t, and $\beta = [\beta_1, \beta_2]$ an hyperparameters of the method, with $\beta_i \in \{0, 1\}$. In this internship, I will mostly use Adam or SGD. With larger and larger networks, along diversity of neural cells, neural networks achieved to be the State-of-the-art (SOTA) in many tasks : classification, computer vision, prediction, NLP ...

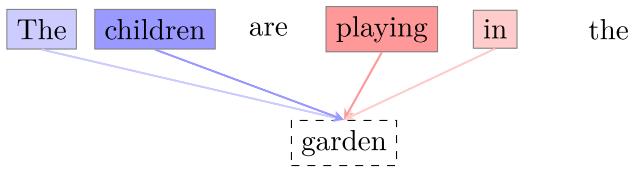

Figure 1.2: Illustration of self attention

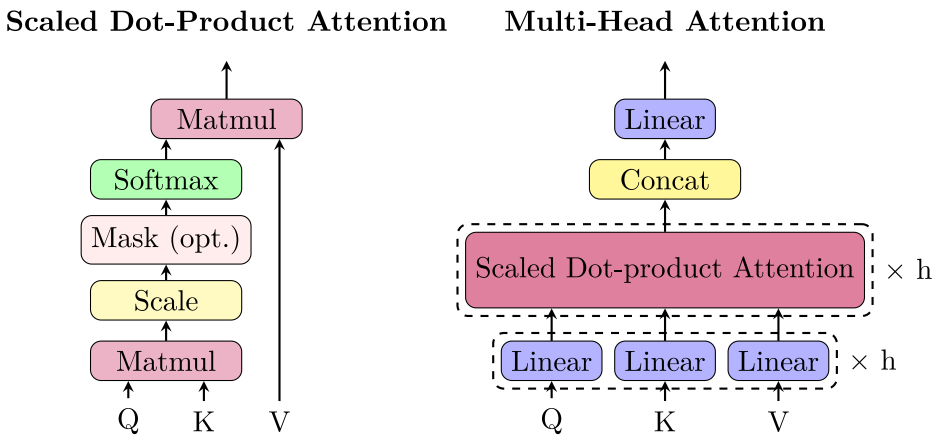

Figure 1.3: Multi-Head Attention (MHA) illustration

1.1.2 Self Attention Mechanism

To understand how LLM works, it's crucial to understand the self attention mechanism. The key feature of LLM is the understanding of the context of a words to perform a prediction, or even a translation. Using Multi-Head Attention (MHA), the *transformers* cells will define the importance of each words for the prediction of the next one. On the example of figure 1.2, to predict the word "garden", the important words are "children" and "playing". The multiplicity of the attention head allow to understand different context with each, like the color on the example.

To perform this feat, Vaswani in article [2] present the scaled dot-product attention, used to build MHA. Transformers use a scaled dot-product attention, shown in figure 1.3, between a queries(Q)/Keys(K) pair, and a value (V) vector. With d_k the dimension of the queries and keys,the attention function can be written as $Attention = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$.

The 3 linear cells are matrices multiplications between an input vector of size d_i to a matrices $W^i \in \mathbb{R}d_{model}.d_i$, with $i \in Q, K, V, O$ (O being the output vector). These matrices are the trainable weights of the Multi-Head Attention, and will be train along the rest of the network. The MHA output can be write as :

$$\begin{gathered} Multihead(Q,K,V) = Concat(head_1, ..., head_h)W^O \\ with\ head_i = Attention(QW_i^Q, KW_i^K, VW_i^V) \end{gathered} \tag{1.3}$$

The softmax function is often used to extract probabilities, since it give values between 0 and 1, and that is take into account every values in order that the sum of softmax value is one. It can be written as : $softmax(x_i) = \dfrac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$.

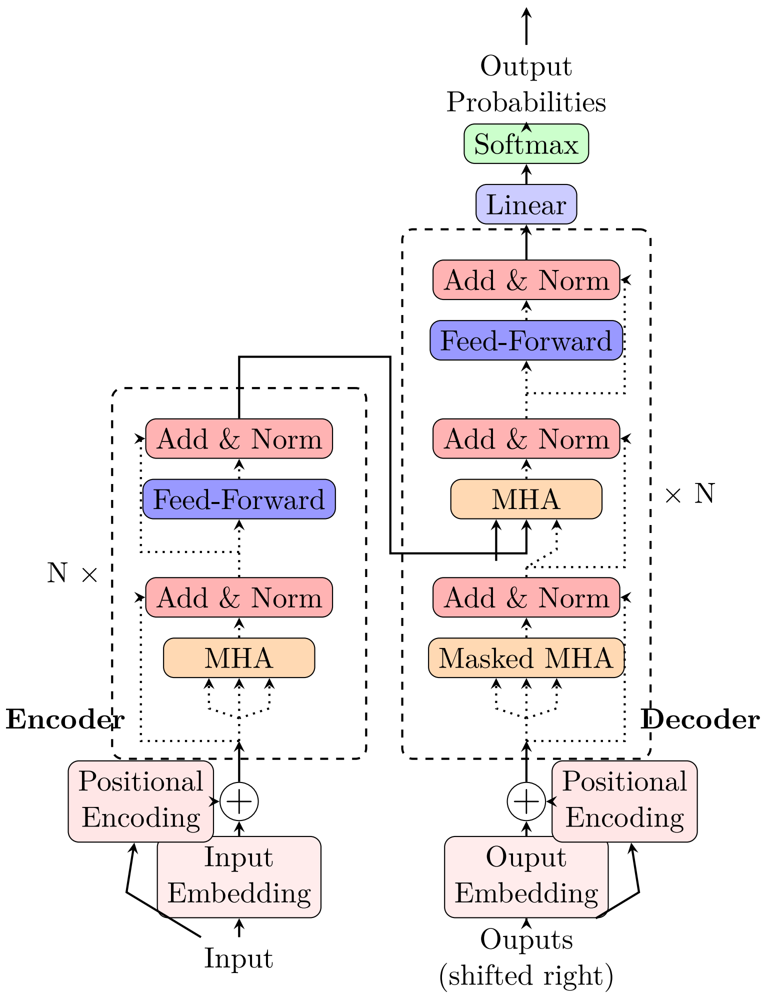

Figure 1.4: Transformers topology (inspired by [2]

1.1.3 LLM Architecture

For many years, NLP problems were approached by statistical and rules models, but they weren't able to capture the context of a whole sentence to generate a word. In 2016, Google published *Google's Neural Machine Translation System* [**Google_translation_2016**], a Long Short-Term Memory (LTSM) neural network, trained on a big corpus of text datas, making it like the first Large Language Models. This was an important break-through, and a proof-of-concept that the neural networks can be the key of NLP Problems.

Once we looked at the MHA in section 1.1.2, we can take a step back and look at the whole transformer architecture, figure **??**. The first step of it is the Embedding, transforming words and sentences into token based on statistical distribution. Along with it, the positional encoding allow the model to keep the position of the token inside the model. A sinusoidal function is used for this, to stay on a relative position and not an absolute.

At left is the **Encoder**, a stack of $N = 6$ identical layers, made of two sub-layer. One is the MHA, and the other is a standard Feed-Forward Layer. Normalisation layer are added between each layers, to avoid exploding gradients. At the left is the **Decoder**, also a stack of $N = 6$ layers. In addition of the two sub-layer of the encoder, the encoder add a second MHA, using the outputs of the encoder and the output of the first decoder's MHA.The masking of the first decoder's MHA is used to only take into account previous word, and not generate word based on next words, since it won't be possible for text generation.

After attention cells, a linear transformation (like inside the MHA) and a softmax are used, in order to have probabilities output of the right dimension size. With this, the model can generate words based on the highest probability of generation.

1.1.4 Fine-Tuning

As of today, the training of LLM is split is two phases : the pre-training and the fine-tuning. The **Pre-training** is computationally very expensive, and only few companies can made one from scratch (OpenAI, Meta, Mistral ...). It also need an enormous corpus of data, often kept hidden from public. Model after pre-training are called Generative Pre-Trained (GPT) model or foundation model. At this point, LLM are able to answer a prompt correctly, with a general amount of knowledge, but not to excel in a specific task.

Aspect	**Pre-Training**	**Fine-Tuning**
Objective	General-purpose learning	Task/domain adaptation
Dataset	Large, diverse	Small, specific
Scale	High resource demand	Relatively efficient
Duration	Weeks to months	Hours to days

Table 1.1: Comparison Between Pre-Training and Fine-Tuning for LLMs

After pre-training, the next stage, known as **Fine-tuning**, is a lighter but equally crucial step that adapts the general knowledge in the model to perform well on specialized tasks. Fine-tuning is often applied to refine the LLM responses by training it on domain-specific datasets or to improve its performance on specific tasks, such as medical diagnosis, legal document analysis, or customer service automation. Research, including studies like [3], suggests that fine-tuning not only helps with task-specific adaptation but also enhances the model's generalization abilities. This process enables the model to transfer the foundational knowledge it learned in pre-training more effectively across a variety of prompts, making it more robust and adaptable. The table 1.1 summarize difference between fine-tuning and pre-training.

Instruction tuning is the process of fine-tuning a pre-trained LLM using datasets composed of instruction-response pairs. The goal is to enhance the model's ability to follow natural language instructions effectively. This involves training the model to generate precise, contextually relevant, and human-aligned responses to a variety of prompts. Instruction tuning often uses datasets containing diverse tasks and instructions, enabling the model to generalize across different domains.

Parameter Efficient Fine-Tuning (PEFT)

PEFT can be defined as the set of techniques used to reduce the cost of fine-tuning. Since fine-tuning aims to be affordable, and usable on consumer-grade hardware, methods must be computation efficient. 2 approachs are used for this : Low Rank Adaptation (LoRA) and Adapter layers.

The **LoRA approach** is based on the LoRA article [4], and includes all methods derived from this : QLora, Lora+... LoRA use Rank Factorization of a matrix, to save the weight matrix

W with 2 reduced weight matrices A and B with $W = B.A$, with $W \in \mathbb{R}^{n*p}, A \in \mathbb{R}^{r*p}, B \in \mathbb{R}^{n*r}$, r being the rank of the matrices factorization. Fine-tuning with LoRA consist of training only theses matrices, and then merge them to the model, resulting in equation 1.4

$$\begin{aligned} W = W_0 + \Delta W = W_0 + B.A \\ s.t. \quad W, W_0, \Delta W \in \mathbb{R}^{n*p}, \\ A \in \mathbb{R}^{r*p} \text{ and } B \in \mathbb{R}^{n*r} \end{aligned} \tag{1.4}$$

The pros of this is the inference that aren't penalized since the model is composed of the same number of weights after the merging. Other pro it that multiple fine-tuning can be effected and they can then be merged when it's needed from a same base, saving only the low-rank version of the weights. One con is that it add hyperparameters, but it will be tackled in nexts sections.

The **Adapter layer** approach is to add an feed-forward layer between two layers and train them to efficiently enhance the model capabilities. The default of using this is that it's increasing the inference for generating predictions, one a the key metrics of LLM.

In the litterature, somes articles before the emergence of LLM are using the fine-tuning of hyperparameters like Hyper-Parameter Optimization (HPO), but in this work, it will solely mean the second phase of LLM training.

1.1.5 Review and taxonomy of LLMs

Exhaustive review of LLMs can be found, like the article [**review_llm**], with details on training, datasets, architectures... On this part, I will focus on key concept and details needed to achieve sufficient understanding for this report.

LLMs Taxonomy

For this taxonomy, the criteria chosen is the downstream tasks, and the corresponding part of the Transformers Architecture used for this.
Encoder-only : LLMs using only the encoder side of the *Transformers* are used to do text analysis or word classification (i.e. extract noun or verb of a sentence). The most famous one is BERT[5] or its variation, an open source model by Google
Decoder-only : these models are used for generative tasks, using prompts to lead it's generation. Generative Pre-Trained (GPT) model, with it's web service ChatGPT [**ChatGPT**] is a decoder-only model, and has strongly contribute to the renowned of the LLMs. We can also cite Llama [**dubey2024llama**] family models, open-source foundation models.
Encoder-Decoder : model using standard Transformers are mainly used for translation, one of the first aim of NLP model, or summarize a text. BART or T5 models are fairly known model for this.

The relevance of this taxonomy come from the compatibility of different optimization method with these three types of models. Different types of models won't be used and train in the same way, and won't have the same hyperparameters.

LLMs review

In this part, I will briefly expose open issues concerning LLMs :
Debiaising data corpus: generative AI base it's representation on dataset, so it tend to reproduce the biaises of the corpus. For example, if we ask ChatGPT to generate 10 name of engineer, we have low probability of having parity.
Interpretability : using neural network in general, but especially LLMs, we can't explain why it works, or why a specific generation happens, so it has limited use.
Efficiency and energy consumption : the result-oriented search tend to only focus on accuracy, rather than energy consumption for a given result. This lead to an over and over-increase of the verticale networks size, and corresponding energy consumption to train and deploy it.
Hallucinations[**chatgpt_hallucinations**] **:** when asking something to the model that does not have the answer, it tend to create a false answer from scratch, and be confident on it. It lower the confidence on the generation.

1.2 Auto-DNN

Automated Deep Neural Networks (Auto-DNN), as defined in [6], refers to the automation of the design and optimization of deep neural network models. This concept is linked with Automated Machine Learning (Auto-ML), but focus solely on DNN. The contribution of Auto-DNN is multiples :

- Excluding humain of the loop, to find new solutions
- Ease the deployment of DNN models, to lessen the needs of expertise
- Ensuring reliability and performance of the solution, with a data-driven approach

Two populars problem of this fields emerged in the last years : **Neural Architecture Search (NAS)** and **Hyper-Parameter Optimization (HPO)**. In this part, I will briefly expose the general formulation of the Auto-DNN problem, and then focus on presenting NAS and HPO specificities.

1.2.1 Problem Formulation

The general Auto-DNN optimization problem can be defined, like article [6], with a quadruplet $\alpha = (V, E, \lambda_V, \lambda_\alpha)$, where V is a set of nodes denoting the neurons, E is a set of edges (i.e. connections) between neurons, λ_V the feature set of operations and λ_α the optimization features set of the DNN. Given the space of all datasets D, the space of models M, and the search space of architectures A.

A first optimization problem can be defined as the training of the DNN, following the objective function $\Theta : A \times D \to M$. The training is then defined as equation 1.5 with L representing the loss function, and d_{train} a subset of an input dataset d.

$$\Theta(\alpha, d) = \text{arg min}_{m_a \in M_a} L(m_a, d_{train}) \tag{1.5}$$

Following the first optimization problem, the second is expressed as the Auto-DNN problem, with equation 1.6. In this equation, f is the objective function, often the negative loss function or the accuracy.

$$a^* = \arg\max_{a \in A} f(\Theta(a, d_{train}), d_{valid} = \arg\max_{a \in A} f(a). \tag{1.6}$$

The Auto-DNN problem is characterized, is the worst case, by theses properties :

- **Mixed Variable-size Optimization Problem (MVOP)** : Variables can be continuous (e.g. learning rate), discrete ordinal (e.g number of layers or neurons) or discrete categorical (e.g. type of activation function). The search space of the problem contains conditionnality, i.e. the size of a variables may depend on other variable (e.g. the number of neurons depend on the number of layers). These properties requires specific optimization methods, or conversion of variables.
- **Expensive black-box objective function :** this problem is a black-box function, i.e. the function cannot be analytically formulated, and so is derivative-free. The evaluation of a solution can take minutes to days, and even days, constraining the number of evaluations. These properties constrains optimization algorithms.

Auto-DNN is an extremely broad field, and it's important to define precisely the specific problem we are treating. Following the notation of article [7] about NAS problems, theses problems are structured according to three fields : Search space, Search strategy and Performances estimation strategy. The **search space** consists of all variables, and theirs properties (range, type ...). The **search strategy** is about the optimization algorithms, defined more thoroughly in section 1.2.4. The **performances estimation strategy** is the definition of the loss function as expressed in equation 1.5, and all linked attributes (e.g. datasets choice). This part can also includes approach like multi-fidelity, modifying the evaluation along iterations.

1.2.2 Neural Architecture Search (NAS)

The NAS problem is a sub-problem of the Auto-DNN problem, where the search space is the topology G as defined in the preceding section. For an exhaustive survey of NAS, one can refer to the article by T. Elsken, *Neural Architecture Search: A Survey* [7]. Figure 1.5 presents the generic workflow of NAS. My contribution in this part focuses on NAS applied to LLM. Due to computational demands, the problem can be addressed using two approaches:

- **Building from Scratch**:
  A classical approach to NAS involves building the architecture from scratch. However, the primary drawback of this method is the significant computational cost, making it accessible only to a limited number of organizations. A remarkable example of this is Google's development of the **AutoBERT-Zero** model [8]. This approach involves discovering an entirely new topology, including new transformer-based structures.
  To mitigate the computational cost, one can constrain the search space. For instance, the number of layers can be fixed, with modifications made to their internal components, or pre-built layers can be fixed, allowing experimentation with their arrangement.
- **Pruning**:
  Pruning involves reducing the size of a model by selecting parts of the topology while aiming to minimize performance degradation. A notable example of this is presented in [9], which uses **weight sharing** methods, such as the approach proposed in [10].

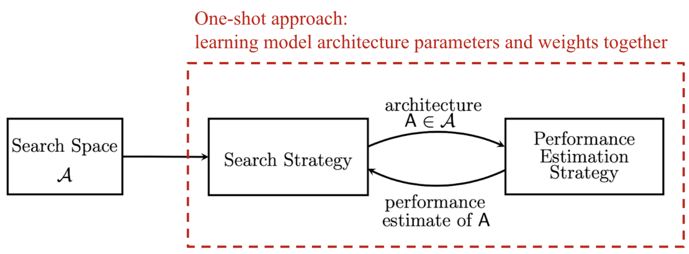
One-shot approach:
learning model architecture parameters and weights together
Search Space $\mathcal{A}$
Search Strategy
architecture $A \in \mathcal{A}$
Performance Estimation Strategy
performance estimate of A

Figure 1.5: Neural Architecture Search Workflow

The methods to solve NAS for LLM are quite diverse. Some methods, such as [**XX**], encode topology into a continuous space, broadening the range of possible optimization techniques. Further details will be addressed in Section 1.2.4, but it is worth mentioning derivative-based methods like [11], which apply such techniques to NAS.

1.2.3 Hyper-parameter optimization

Like NAS, HPO can be defined as a sub-field of Auto-ML, even if HPO by itself is tackled since the 1990s [**hutter2019automated**]. In Deep Neural Networks (DNN), hyperparameters can be defined as configuration settings that govern the structure of the networks and the process of training. As opposed to parameters (or weights), hyperparameters are not learning directly from the data during the training. They are typically set before training begins and remain fixed throughout the process.

Most of the time, hyperparameters are chosen by humans, w.r.t. their expertise. HPO is the process of choosing the best hyperparameters for a specific problem (a quadruplet a as defined in 1.2.1). However, manually selecting hyperparameters is often inefficient and prone to suboptimal configurations, especially as models grow in complexity. Automated HPO methods aim to address these challenges by systematically exploring the search space to identify configurations that maximize performance or minimize error for a given task.

The significance of HPO grows with the increasing complexity of modern DNN architectures. As models become larger and datasets more diverse, the choice of hyperparameters can significantly impact both model accuracy and computational efficiency. Moreover, HPO plays a critical role in enabling the deployment of models in resource-constrained environments, where trade-offs between accuracy and efficiency must be carefully balanced. Future sections will detail methodologies and frameworks designed to tackle HPO effectively in various contexts.

1.2.4 Optimization Algorithms taxonomy

Global optimization refers to the field of mathematical and computational methods designed to find the best solution to a problem within a defined domain, particularly when the objective function is complex, non-linear, or multi-modal. Traditional methods often struggled with problems involving multiple local optima or discontinuities.

To address these challenges, the field has evolved to include more robust techniques, such as metaheuristics, surrogate modeling, and hybrid approaches.

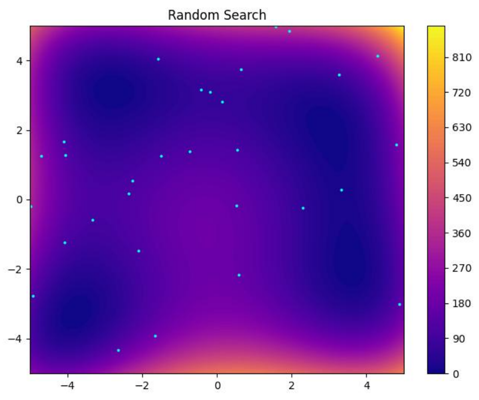

Figure 1.6: Random and Grid Search on Himmelblau 2D function

These methods aim to efficiently navigate vast and complex search spaces, balancing the trade-off between exploration (searching new regions) and exploitation (refining known good solutions). In the context of hyperparameter optimization (HPO), global optimization plays a crucial role in systematically identifying configurations that maximize model performance while minimizing computational costs. The subsequent sections categorize and detail these optimization methods.
ADD A FIGURE BY PARAGRAPH WITH A 1D OR 2D FUNCTION AND WRITE HERE THE FUNCTION.

Exploratory Methods

Basic approaches such as Grid Search (GS) (Grid Search) and Random Search (RS) (Random Search) provide straightforward solutions for hyperparameter optimization (HPO). GS systematically evaluates all possible combinations of hyperparameters within a predefined grid, making it simple to implement and interpret. However, this approach becomes computationally intractable in high-dimensional hyperparameter spaces due to the exponential increase in the number of configurations. On the other hand, RS selects hyperparameters randomly from the search space, offering a more scalable alternative to GS while maintaining simplicity. Despite its limitations, RS is often used as a baseline or guideline to compare with more sophisticated optimization methods.

Metaheuristic Approaches

Metaheuristic methods, including simulated annealing and evolutionary algorithms (EA), are designed to search more efficiently within large hyperparameter spaces by mimicking natural or physical processes. Simulated annealing is inspired by the cooling process of metals, where the algorithm explores the search space by gradually reducing the probability of accepting worse solutions. This allows it to escape local optima and converge to a globally optimal solution over time. Evolutionary algorithms, on the other hand, draw inspiration from biological evolution, employing operators such as mutation, crossover, and selection to iteratively improve a population of candidate solutions. These approaches are particularly useful for complex, non-convex hyperparameter spaces where simpler methods like GS or RS might struggle.

Partition-Based Optimization

Partition-based optimization methods aim to divide the hyperparameter search space into smaller, manageable subspaces, focusing the search on the most promising regions. These

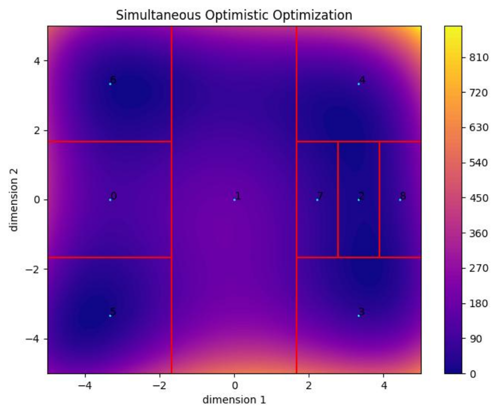

Figure 1.7: Example of a Partition Based Optimization on 2D himmelblau

techniques dynamically refine the partitions based on previous evaluations, progressively narrowing down the search space. Examples include methods like Hyperband, which uses successive halving to allocate resources adaptively across configurations, and bandit-based approaches that efficiently balance exploration and exploitation. Partition-based methods are especially effective when dealing with large search spaces and limited computational budgets.

Surrogate-Model Based Optimization

Surrogate models, such as those employed in Bayesian Optimization (BO), offer a powerful framework for hyperparameter optimization. Instead of directly evaluating the objective function for every configuration, which can be computationally expensive, surrogate models approximate the objective function based on a limited number of evaluations. Gaussian Processes (GP), commonly used in BO, provide a probabilistic model of the objective function, capturing uncertainty and guiding the search toward the most promising regions of the hyperparameter space. Other surrogate models, such as tree-structured Parzen estimators (TPE), are also effective in optimizing complex, high-dimensional spaces. These methods are particularly advantageous when each evaluation of the objective function (e.g., training a machine learning model) is time-consuming or resource-intensive.

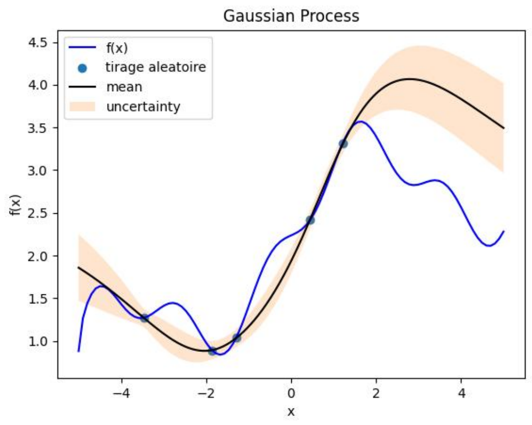

Figure 1.8: Gaussian Process example

1.2.5 Parallel Optimization and High Performance Computing

Parallel computation and high-performance computing (HPC) are fundamental to advancing artificial intelligence (AI). The complexity of optimization problems and resource demands of AI models necessitate parallel architectures, from multicore processors to distributed systems. Techniques like algorithmic and solution-level parallelism enable efficient exploration of massive search spaces, speeding up tasks such as evolutionary computation and deep learning optimization. Specialized hardware, such as GPUs and FPGAs, further accelerates matrix operations critical to AI workloads.

HPC platforms connect distributed clusters and grids, enabling scalable solutions to real-world challenges, including large-scale neural network training and hyperparameter optimization. NumPEx, through its Exa-MA (Methods and Algorithms for Exascale) subprogram, addresses computational challenges of exascale systems by developing scalable algorithms and architectures. Your work aligns with Exa-MA's goals of advancing energy-efficient methods and leveraging HPC for predictive modeling, strengthening the synergy between AI innovation and exascale computing.

1.3 LLMs application to manufacturing context

To link this internship to my engineering curriculum, I will explore the application for LLMs to manufacturing. The application can be split in three : quality control, Supply-chain management, predictive maintenance. Among these application, the two key-point of LLMs are the extensive reasoning, that can be used for prediction or analysis, and the performance in NLP problems, proving itself to be able to simply interact with any operators.

1.3.1 LLMs-based quality control

Quality control can be defined as "procedure or set of procedures intended to ensure that a manufactured product or performed service adheres to a defined set of quality criteria or meets the requirements of the client or customer" [**whatis_QC**]. In this field, the last decades was really interesting from a data management point of view, since it's was the emergence of a lot of data collection : product measurement along manufacturing executive system (MES), customer feedbacks... Since a lot of unstructured data is collected, deep learning can be a way to use all of these to manage quality control. We can extract few specific ways :

- Automated Inspection : LLM are more and more multimodals, and it's possible to achieve computer vision along the reasoning capacities. It's now possible to go further and use camera to control every products on a conveyor.
- Customer Feedback : LLM can be used to summarize and prioritize feedback, to be able to use precise insight on the manufacturing process.

1.3.2 LLMs-based supply chain management

Since decades, **demand forecast** is a hot field of supply chain management, with many methods considering means and data collection of the company. LLMs, and in specific time-LLM [**timeLLM_2024**], achieve a new step in automated reasoning, and to the corpus of data taken into account. From the first method using only historical sales, it's now possible to use a wider historical data, like climate, economical situation ... LLM can also be used for **supplier evaluation**. When considering a relationship with a supplier, many information can be used to characterise it : prices, delay, defaults, reactivity ... And LLM can summarize all of this to provide insight for the choice of a supplier on a specific project.

1.3.3 LLMs-based predictive maintenance

By combining keys aspect of automated inspection and demand forecast, LLM could be a great asset in **maintenance scheduling**. One key point of these model is that they could use every possible inputs : machine logs, humain report, captures... With this, they could reach the best of humain (multi-modality) and machine (long, tedious tasks) to achieve state-of-the-art performance on these subject. More over, LLMs can be used for **root cause analysis**, to find the root of anomaly or failure and reduce the down time of the manufacture.

1.4 Search problematic

My first two weeks were focus on understanding the context of the internship, mainly by reading articles, and working on extracting only one search problematic. To do this, my main concerns were :

- Feasibility : 24 weeks can be short for a too long research plan, all the more with long experiments. I need to be able to understand and assess the problem, try and implement methods.
- Costs : can't buy or build supercomputer just for this, and the cost of long computing of the resource can't be ignored
- Research team expertise : the team is firstly optimization oriented, so problematic only oriented to LLMs won't be the priority

After a first week, the possible fields was narrowed to two tracks. The first one is based on article [9], and involve the pruning of a large pre-trained model, in order to reduce latency without losing to much accuracy. The other one [12] is Hyper-parameter Optimization (HPO) applied to Instruction tuning.

Eventually, after reflection and discussion with my tutor, we choose to address the HPO of LLM fine-tuning. This choice was made to reduce the uncertainty on an already very exploratory fields, since HPO was already tackled on different type of NN.

The objective of my internship is then to work on HPO methods like apply to LLM fine-tuning. This work includes :

- **Reproduce the objective function :** on a given library, implement the black box function of training and evaluate a LLM.
- **Definition of the search space :** balancing between curse of dimensionnality and research ambition, find pertinent hyperparameters to work with, and define theirs properties.
- **Selection and Implementation of Optimization algorithm :** considering littera-

ture and available frameworks, choose relevant algorithms and implement it to the problem

- **Make experiments :** following a rigorous protocol, make experiments to determine the effects of the optimization, and analysis it
- **Formalize the contribution :** With this report and a scientific article, explicit the contribution, and formalize for furthers works

Chapter 2
Methodology

Everyone by now presumably knows about the danger of premature optimization. I think we should be just as worried about premature design - designing too early what a program should do.

Paul Graham

The methodology is a cornerstone of any research or project, providing a structured framework to achieve objectives systematically and effectively. It ensures clarity, reproducibility, and reliability by defining the steps, tools, and techniques used to address specific problems. A well-defined methodology not only aligns the research process with its goals but also facilitates critical evaluation by external audiences, allowing them to assess the validity and generalization of the results. In the context of this work, the chosen methodology was pivotal in navigating complex challenges, optimizing processes, and ensuring that outcomes are both credible and relevant.

To ensure my sincere approach, and contribute to open-source domain, all the code of this report is readable on my github account[2]. It include every code snippet used in this report (like figure 1.7), every experiments of chapter 3 and all implementation of HPO.

In this chapter, I will talk about the contextualization in academic literature, then tackle the elaboration of the blackbox function. The definition of the search space being one of the most crucial step in global optimization, the section **??** will focus on this. After this preliminary work, we will enter the core of this report : optimization algorithms. A section about experimental setup, to explore resource and scientific integrity, will precede the conclusion section, approaching insight about the realization of this part.

2.1 A Literature-Based Approach

In industrial field of works, the goal is to be better than competitor, or at least be better than the past of the company. In research fields, a contribution must aims to be better than existing, at least by one facet. In order to do this, the first step of every research project is to make an exhaustive bibliography of the domain, to understand what's already done, and what could be the contribution of the project.

Chapter 1 was the result of a first stage of bibliography, to define what's the context of this internship. With this, we have insights and contexts about DNN,LLM,fine-tuning and PEFT, and a first look at global optimization fields. In this chapter, a complementary approach will be done about specific optimization algorithm, frameworks and implementation specific details.

[2]link : https://github.com/Kiwy3/LLM_HPO_BO___st30

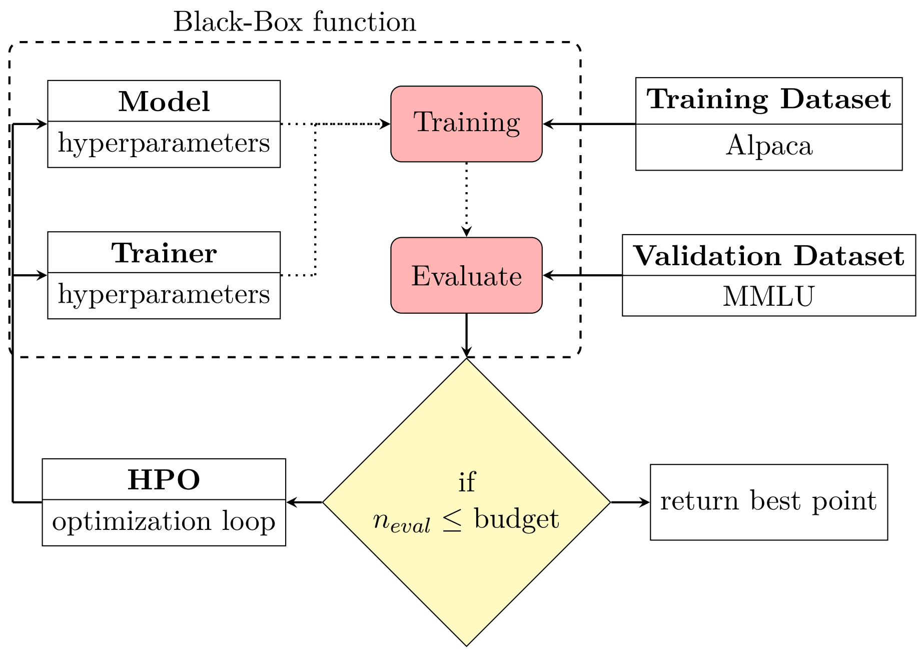

Figure 2.1: HPO workflow

At the beginning of this internship, I started my bibliography using few articles that my tutor send me, for a first look of the subject. From theses articles, I jumped to referenced articles until I started to make a loop between articles. It allow me to find fundational article like articles [2, 6], establishing the core of the domain, and reviews like article [7, 6], allowing to understand a global context and finding a way to classify what I read before.

To manage my bibliography, in a first time I used Notion App[1] to make a table for my bibliography, with papers charateristics (title, authors, year ...), an export of bibtex from original site and my notes. The table can be found on this link. When I started writing my article, I thought that it's wasn't pratical to copy bibtex export one by one, and I looked at others tools to manage this. It's how I found Zotero[2], with many options to ease my life like collecting article from web with only one click, and export a collection.

2.2 Blackbox Elaboration

My internship can be seen as global optimization applied to an expensive blackbox function. A blackbox function is a process that receive an input (here a set of hyperparameters), and return one (or multiple) value(s) (here the accuracy). The blackbox process here is described by figure 2.1. This process start by the fine-tuning of the model, using training dataset, and then evaluating the model, using the validation dataset. Next sections will explore in details the action box of figure 2.1.

[1]https://www.notion.so
[2]link : https://www.zotero.org/

2.2.1 Fine-Tuning of the Model

Choose the model : TinyLlama and then LlaMa 3.2-3B Choose the dataset

For the fine-tuning, I use LoRA as defined in section 1.1.4

Add lora figure

Fine tuning process is like general nn training, following algo X

Add algo of fine tuning

For the implementation : model from hf, class from litgpt, automation from lightning, with hooks

2.2.2 Evaluation of the model

To evaluate, two ways : loss or accuracy => why I choose acc, and even acc_norm => fine tune model are zero shot learners

generic evaluation with algo X

I choose MMLU/HELLASWAG

Only one for HPO, but can eval on others to look at overfitting

implementation with lm_eval

Final result (eval + training) : a class

class diagramm ???

2.3 Search Space Definition

2.4 Optimization

To address the challenges of optimizing the black-box function, this research introduces two complementary approaches: Bayesian Optimization (BO) and a Partition-based method. BO is employed for its ability to efficiently navigate continuous hyperparameter spaces by balancing exploration and exploitation through surrogate modeling and acquisition functions. The Partition-based method divides the search space into regions, enabling parallel optimization and reducing redundancy in evaluations. By combining these methods, the framework achieves robust performance across diverse benchmarks. This integration is further enhanced by incorporating techniques such as multi-fidelity optimization and FlashAttention to improve computational efficiency and scalability.

2.5 experiments setup

2.6 Difficulties

Several challenges were encountered during the development and implementation of this optimization framework. The foremost difficulty is the high computational expense of evaluating LLMs, which necessitates the careful allocation of resources and the adoption of efficient evaluation strategies. Handling mixed-type hyperparameters, particularly the interplay between continuous and discrete variables, posed additional complexities. Existing optimization techniques often struggle with these mixed spaces, requiring innovative solutions such as relaxation and partitioning to ensure convergence. Finally, ensuring the generalizability of the fine-tuning results across different benchmarks and datasets proved challenging, as model performance is highly dependent on task-specific characteristics and dataset quality.

Chapter 3
Results

This chapter presents the outcomes of the proposed methods described in the preceding chapter and provides a reflective analysis of the findings. Additionally, it explores how these results contribute to the broader scientific community, including efforts toward publication. The chapter concludes by outlining potential directions for extending this research in future work.

3.1 Experiment Results

The experimental results from Chapter 3 validate the efficacy of the proposed approaches for hyperparameter optimization (HPO) in fine-tuning Large Language Models (LLMs). Key performance metrics, including accuracy on benchmark datasets (e.g., MMLU, GLUE), computational efficiency, and scalability, demonstrate that the hybrid approach combining Bayesian Optimization (BO) and Partition-based methods achieves competitive results while significantly reducing evaluation costs.

The analysis highlights the following:

- Bayesian Optimization effectively balances exploration and exploitation, enabling faster convergence in continuous hyperparameter spaces.
- The Partition-based method complements BO by parallelizing the search process, enhancing scalability for larger hyperparameter spaces.
- Techniques such as Low-Rank Adaptation (LoRA) and multi-fidelity evaluations improve computational efficiency without compromising performance.

Tables and figures (e.g., Table XX, Figure YY) illustrate these findings, providing a comparative evaluation of the proposed methods against baseline approaches such as grid search and random search. Although the results are promising, limitations are noted, including sensitivity to hyperparameter initialization and computational overhead in scenarios with extensive partitioning schemes.

3.2 Article Publication

The aforementioned results have led to the drafting of a research article aimed at disseminating the findings within the academic community. Publishing the article serves as a means of contributing to the broader field of optimization and machine learning research.

The written article has been submitted to the International Conference on Optimization & Learning (OLA2025). As of the submission of this report, the paper is under review, and the outcome is awaited.

3.3 If This Work Were to Be Continued

While this research represents significant progress, several avenues remain for future exploration:

- **Extending the Search Space:** Expand the range of hyperparameters to include advanced configurations, such as specialized LoRA parameters and task-specific pre-training strategies.
- **Integrating Advanced Techniques:** Investigate the use of cutting-edge approaches, such as reinforcement learning-based optimization or meta-learning, to improve generalization across diverse datasets.
- **Scaling Beyond Current Limits:** Enhance the framework to accommodate even larger LLMs and datasets by leveraging distributed computing infrastructures and more sophisticated partitioning strategies.

These directions offer a clear path for further building upon the contributions of this work, ensuring its continued relevance in addressing emerging challenges in the field of LLM fine-tuning and optimization.

Chapter 4
Reflexion and Prospection

4.1 Generalisation of the work

pin the expensive function optimization, and generalize with other ones.

4.2 an industrial context

4.3 an intership, part of a bigger cursus

explain how this internship complement my formation, and is useful for my following career.

Conclusion et perspectives

summary

explain the key results, and why it's important

end-of-studies

explain why this internship was perfect for my cursus

References

[1] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization.* arXiv:1412.6980. Jan. 2017. DOI: 10.48550/arXiv.1412.6980. URL: http://arxiv.org/abs/1412.6980 (visited on 11/20/2024).

[2] Ashish Vaswani et al. "Attention is All you Need". In: *Advances in Neural Information Processing Systems.* Vol. 30. Curran Associates, Inc., 2017. URL: https://papers.nips.cc/paper_files/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html (visited on 11/20/2024).

[3] Jason Wei et al. *Finetuned Language Models Are Zero-Shot Learners.* arXiv:2109.01652 [cs]. Feb. 2022. URL: http://arxiv.org/abs/2109.01652 (visited on 11/20/2024).

[4] Edward J. Hu et al. *LoRA: Low-Rank Adaptation of Large Language Models.* arXiv:2106.09685. Oct. 2021. DOI: 10.48550/arXiv.2106.09685. URL: http://arxiv.org/abs/2106.09685 (visited on 11/20/2024).

[5] Jacob Devlin et al. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* arXiv:1810.04805. May 2019. DOI: 10.48550/arXiv.1810.04805. URL: http://arxiv.org/abs/1810.04805 (visited on 11/20/2024).

[6] El-Ghazali Talbi. "Automated Design of Deep Neural Networks: A Survey and Unified Taxonomy". In: *ACM Comput. Surv.* 54.2 (Mar. 2021), 34:1–34:37. ISSN: 0360-0300. DOI: 10.1145/3439730. URL: https://dl.acm.org/doi/10.1145/3439730 (visited on 11/20/2024).

[7] Thomas Elsken, Jan Hendrik Metzen, and Frank Hutter. *Neural Architecture Search: A Survey.* en. arXiv:1808.05377 [stat]. Apr. 2019. URL: http://arxiv.org/abs/1808.05377 (visited on 11/20/2024).

[8] Jiahui Gao et al. "AutoBERT-Zero: Evolving BERT Backbone from Scratch". en. In: *Proceedings of the AAAI Conference on Artificial Intelligence* 36.10 (June 2022). Number: 10, pp. 10663–10671. ISSN: 2374-3468. DOI: 10.1609/aaai.v36i10.21311. URL: https://ojs.aaai.org/index.php/AAAI/article/view/21311 (visited on 11/20/2024).

[9] Aaron Klein et al. "Structural Pruning of Large Language Models via Neural Architecture Search". en. In: (Oct. 2023). URL: https://openreview.net/forum?id=VAwgL8kPvr (visited on 11/20/2024).

[10] Hieu Pham et al. *Efficient Neural Architecture Search via Parameter Sharing.* arXiv:1802.03268. Feb. 2018. DOI: 10.48550/arXiv.1802.03268. URL: http://arxiv.org/abs/1802.03268 (visited on 11/25/2024).

[11] Hanxiao Liu, Karen Simonyan, and Yiming Yang. *DARTS: Differentiable Architecture Search.* arXiv:1806.09055. Apr. 2019. DOI: 10.48550/arXiv.1806.09055. URL: http://arxiv.org/abs/1806.09055 (visited on 11/20/2024).

[12] Christophe Tribes et al. *Hyperparameter Optimization for Large Language Model Instruction-Tuning.* arXiv:2312.00949. Jan. 2024. DOI: 10.48550/arXiv.2312.00949. URL: http://arxiv.org/abs/2312.00949 (visited on 11/20/2024).