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**

utt
UNIVERSITÉ DE TECHNOLOGIE
TROYES

Inria

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.

Summary

Tables and Figures

Algorithms

Glossary and Acronyms

Introduction **1**

1 Company's presentation **2**
1.1 INRIA history . . . 2
1.2 INRIA center of Lille University . . . 3
1.3 BONUS team . . . 3

2 Subject Definition **4**
2.1 Large Langage Models (LLM) . . . 4
2.2 Auto-DNN . . . 11
2.3 LLMs application to manufacturing context . . . 17
2.4 Search problematic . . . 18

3 Methodology **19**
3.1 A Literature-Based Approach . . . 19
3.2 Blackbox Elaboration . . . 20
3.3 Search Space Definition . . . 22
3.4 Optimization Algorithms . . . 23
3.5 Concrete Implementation . . . 27
3.6 experiments setup . . . 28
3.7 Difficulties . . . 28

4 Results **29**
4.1 Experiment Results . . . 29
4.2 Article Publication . . . 29
4.3 If This Work Were to Be Continued . . . 30

5 Reflexion and Prospection **31**
5.1 Generalisation of the work . . . 31
5.2 an industrial context . . . 31
5.3 an intership, part of a bigger cursus . . . 31

Conclusion et perspectives **32**

References

A Annexes **II**
A.1 Algorithm . . . II

A.2 Captures d'écran III
A.3 Formules chimiques III
A.4 Extraits de code source III

List of Tables

2.1 Comparison Between Pre-Training and Fine-Tuning for LLMs 8

3.1 Summary of Hyperparameter Search Space 23

List of Figures

2.1 Illustration of an artifical neuron 5
2.2 Illustration of self attention 6
2.3 Multi-Head Attention (MHA) illustration 6
2.4 Transformers topology (inspired by [41] 7
2.5 Pre-Training and Fine-Tuning Framework 8
2.6 illustration of adapter layer 9
2.7 illustration of lora layer 10
2.8 Neural Architecture Search Workflow 13
2.9 Exploratory Methods on Himmelblau 2D function 14
2.10 Example of Iterated Local Search on himmelblau 15
2.11 Example of a Partition Based Optimization on 2D himmelblau 15
2.12 Gaussian Process example on $f(x) = \sin(x)^2 + \sqrt{x+5}$ 16

3.1 HPO workflow 20
3.2 Class diagramm of the optimization framework 27

List of Algorithms

3.1 Fine-Tuning(model, hyperparameters) 22

3.2 BO 24

3.3 SOO 25

3.4 BamSOO scoring 26

A.1 BamSOO II

Glossary

fine-tuning 2nd step of LLM training. 4, 8–10, 18–21

HuggingFace Deep Learning Hub with models and datasets. 21, 22
hyperparameter Parameters not learned by the model. 10, 13, 18

instruction tuning Fine-tuning with Instruction and behavior dataset. 9

litgpt PyTorch based framework for training LLM. 21, 22

performances estimation strategy need to define it. 12
pre-training 1st step of training LLM. 9
PyTorch Tensor-based framework for machine learning. 21
PyTorch Lightning automated deep learning training framework. 21, 22

search space need to define it. 12, 13
search strategy need to define it. 12

transformer Neural networks layers type using attention mechanisms. 4–8

Acronyms

Adam Adaptive Moment Estimation. 5, 21
AdamW Adaptive Moment Estimation with Weight Decay. 21
ANN Artificial Neural Networks. 5, 21, 22
Auto-DNN Automated Deep Neural Networks. 4, 11, 12, 16
Auto-ML Automated Machine Learning. 11, 13

BaMSOO Bayesian Multi Scale Optimistic Optimization. 26, 28
BO Bayesian Optimization. 16, 23, 24, 28

CLI Command Line Interface. 21, 22
CNN Convolutional Neural Networks. 5

DNN Deep Neural Networks. 4–6, 11, 13, 16, 19

EA Evolutionnary Algorithms. 5, 14

FDA Fractal Decomposition Algorithm. 15
FLOPS Floating-Point Operations per Second. 3

GA Genetic Algorithms. 15
GP Gaussian Process. 16, 23–28
GPT Generative Pre-Trained. 8, 21
GPU Graphics Processing Unit. 16
GS Grid Search. 14, 15

HPC High Performance Computing. 16
HPO Hyper-Parameter Optimization. 1, 4, 10, 11, 13–15, 18, 19, 22, 23

ILS Iterated Local Search. 15

LCB Lower Confidence Bound. 26–28
LHS Latin Hypercube Sampling. 24
LLM Large Language Models. 1, 4, 6–10, 12, 17–19, 22
LoRA Low Rank Adaptation. 9, 21, 22
LTSM Long Short-Term Memory. 7

MCQ Multi-Choice Question. 22
MCTS Monte Carlo Tree Search. 25
MHA Multi-Head Attention. 6, 8, 21
ML Machine Learning. 16
MLP Multi-Layer Perceptron. 5
MSE Mean-Squared Error. 5
MVOP Mixed Variable-size Optimization Problem. 11

NAS Neural Architecture Search. 4, 11–13

NLP Natural Language Processing. 4, 6, 7
NN Neural Networks. 18

OOP Object Oriented Programming. 27

PBO Partition Based Optimization. 15, 16, 23
PEFT Parameter Efficient Fine-Tuning. 9, 19, 21
POC Proof-of-Concept. 17

QC Quality Control. 17

RS Random Search. 14, 15

SA Simulated Annealing. 14, 15
SCM Supply Chain Management. 17
SGD Stochastic Gradient Descent. 5
SMBO Surrogate-Model Based Optimization. 16, 23, 26
SOO Simultaneous Optimistic Optimization. 15, 23, 25, 26, 28
SOTA State-of-the-art. 6, 21

$\mathcal{UCB}$ Upper Confidence Bound. 26–28

VAE Variable Auto-Encoder. 5

Introduction

From September 09, 2024, to February 21, 2025, I joined the BONUS project-team, Big Optimization aNd Ultra-Scale computing, at the INRIA Center of the University of Lille, under the supervision of Mr. El-Ghazali Talbi. Adding a Master's degree in Systems Optimization and Safety to my Industrial Engineering curriculum, the intrinsic question was : do I want to continue in the research world ? From the point of view of building my career path, this internship is designed to answer this question.

As one of France's historic research institutions, INRIA is a leading figure in information and digital research. It was within the framework of the Priority Research Program and Equipment (PEPR) called Numérique pour l'Exascale (NumPEx), and in particular the Exa-MA axis, that I carried out this internship. Without going into too much detail, the aim of my internship is to pursue research into Neural Architectural Search (NAS), applied to Large Language Models (LLM).

Après une première phase présentant en détail l'INRIA, nous pourrons poursuivre par un focus sur le sujet du stage, et le contexte dans lequel il s'inscrit.

Chapter 1
Company's presentation

No research without action, no action without research

Kurt Lewin

INRIA, National Research Institute for Computer Science and Automation,is one of the leading public institutions involved in academic research in France. Today, more than 3,800 scientists, working in 220 project teams, are involved in digital research at INRIA. Since its creation, INRIA's mission has been to ensure French sovereignty and autonomy in IT-related fields, while transferring knowledge to the industrial world.

INRIA is made up of 10 research centers, spread across France, which work in a dozen areas, including :

- High-performance computing
- Digital Education
- Artificial intelligence
- Digital health
- Data science
- Software

To support this, the institute is developing a large number of partnerships, playing its role as a research vector. These partnerships include, first and foremost university partnerships; the research centers are attached to universities, in order to contribute in the training of tomorrow's scientists. We can also add institutional research partners, such as CNRS or CEA in France, and many others in France, and many others in Europe, which enable us to take on large-scale projects. projects. And last but not least, our industrial partnerships help to keep the Institute going. These range from the 170 start-ups incubated on INRIA premises over the past 20 years, to industry giants such as Microsoft, for example, who collaborate in the joint Microsoft-Research / INRIA joint research center[16]

1.1 INRIA history

INRIA was founded in 1967, under the name "Institut de Recherche en Informatique et Automatique" (IRIA), as part of the *Plan Calcul*[31]. This project, launched by the French government in 1966, was designed to ensure France's autonomy and sovereignty in the field of information technology. A few years later, in the 70s, INRIA led the *Cyclades* project, participating in the networking of computers, and contributing to what would later become the Transmission Control Protocol (TCP).

In 1979, the institute affirmed its commitment to a national structure, and became INRIA, with the opening of centers in Rennes, then Sophia-Antipolis, Nancy and Grenoble. The same dynamic led to the creation of Simulog, the first start-up incubated at INRIA, reaffirming the Institute's commitment to innovation in the broadest sense of the term.

In the early 2000s, INRIA, like the rest of the world, was strongly influenced by the development of the Web and its applications. In particular, INRIA was responsible for the European node of the W3C (World Wide Web Consortium). Since then, INRIA has diversified its research, supporting research into digital health and developing significant expertise in software engineering.

1.2 INRIA center of Lille University

The INRIA center at the University of Lille was born of a partnership between INRIA and the University of Lille, in 2007. It first set up a site in Villeneuve d'Asq [1], close to the scientific city campus, which also houses Polytech Lille and Centrale Lille, then a second site in Lille [2], in the Euratechnologies park. It currently houses 385 employees, including 260 researchers and 50 engineers, divided into 15 project teams. In line with national priorities, while at the same time asserting its specialties, the center focuses on 5 themes:

- Data Science
- Software Engineering
- Cyber Systems Physics
- Digital Health
- Digital Sobriety

1.3 BONUS team

The BONUS team (Big Optimization aNd Ultra Scale computing) is part of the INRIA center at the University of Lille, located on the Lille site, and comprises around 15 people, including 4 researchers. It focuses on large-scale optimization problems, which are characterized by: 1) a large number of problem dimensions 2) the possibility of multi-objective optimization 3) very high solution evaluation costs 4) the need for supercomputers. To address these issues, the team has divided its research into 3 areas:

- Decomposition-based optimization: defining and solving sub-problems to approximate the larger problem
- Optimization supported by machine learning: machine learning, and by extension artificial intelligence, concentrates a set of problems that require optimization.
- Large-scale computation: for these problems, the need for computation is reaching levels that seemed unimaginable just a few years ago. Parallelization and the development of suitable hardware are the key to progress.

INRIA, through its BONUS team, is involved in the PEPR NumPex[28], which aims to enable the development and use of exascale computing[3]. This internship is part of the Exa-MA priority project, which aims to develop mathematical methods and algorithms for translating simulated phenomena into equations.

[1] 40, avenue Halley - 59650 Villeneuve d'Ascq
[2] 172, avenue de Bretagne - 59000 Lille
[3] machines capable of performing at least 10^{18} FLOPS

Chapter 2
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.

2.1 Large Langage Models (LLM)

LLM can be defined as Deep Neural Networks (DNN) 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 2.1.4. To pursue to the next section, I will finish with a light review and taxonomy of LLM.

2.1.1 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 [25]. 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 2.1 show the structure of an artifical neuron, taking input x_i and trainable weights ω_i, including the biais ω_0. The output of the neuron is expressed as : $\hat{y} = f(x, \omega) = \sigma(\sum_{i=1}^{n} x_i\omega_i + \omega_0)$, with $\sigma(.)$ the activation function.

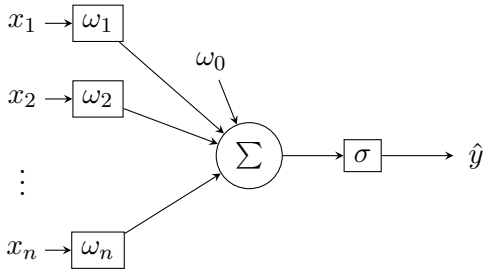


Figure 2.1: Illustration of an artifical neuron

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 to be 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 and called *meta-learning*[14], but were deprecated in favor of gradient-based methods.

With the notation of figure 2.1, and ω being the matrix of all parameters, the training of the ANN can be expressed as equation 2.1. 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[48] or Mean-Squared Error (MSE).

$$\omega \in \arg\min_{\omega \in \mathbb{R}^n} \mathcal{L}(\hat{y}, y) \tag{2.1}$$

The optimization of equation 2.1 is done using gradient descent, and especially Stochastic Gradient Descent (SGD). For each parameter, the gradient is expressed as 2.2. Based on this, the parameter is updated with $\omega_i \leftarrow \omega_i - \eta * \Delta_{\omega_i}$.

$$\Delta_{\omega_i} = \Delta_{\omega_i}(x, y) = \frac{\partial \mathcal{L}(f(x, \omega), y)}{\partial \omega_i} \tag{2.2}$$

To accelerate the gradient descent, some optimization algorithm like Adaptive Moment Estimation (Adam) [20] 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.

From simple Multi-Layer Perceptron (MLP) to more complexe architecture (Convolutional Neural Networks (CNN), Transformers,Variable Auto-Encoder (VAE)...), from dozen to bilions

of parameters, the DNN are became over the years the State-of-the-art (SOTA) in many tasks : classification[47], computer vision[34], prediction [19], NLP[9] etc.

2.1.2 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 2.2, 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 one, like the color on the example.

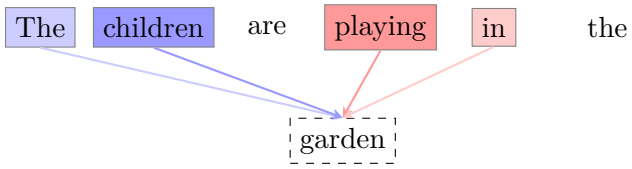


Figure 2.2: Illustration of self attention

To perform this feat, Vaswani in article [41] present the scaled dot-product attention, used to build MHA. Transformers use a scaled dot-product attention, shown in figure 2.3, 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 $Attention = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V$.

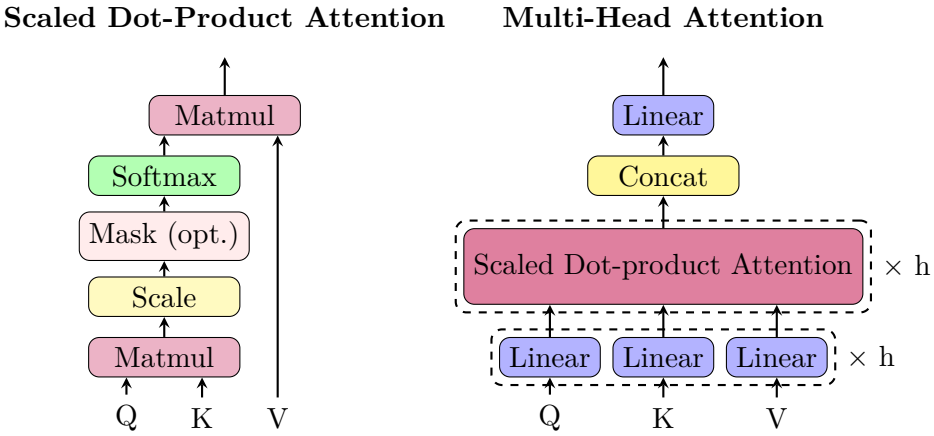


Figure 2.3: Multi-Head Attention (MHA) illustration

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 :

$$\begin{aligned} Multihead(Q,K,V) &= Concat(head_1, ..., head_h)W^O \\ with\ head_i &= Attention(QW_i^Q, KW_i^K, VW_i^V) \end{aligned} \tag{2.3}$$

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 : $softmax(x_i) = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}}$.

2.1.3 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* [45], 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 neural networks can be the key of NLP Problems.

To perform further, what NLP needed was the ability to understand a context, to predict or understand a sentence. To do that, the Transformer architecture was published in 2017 [41], and was using the self attention mechanism presented in section 2.1.2.

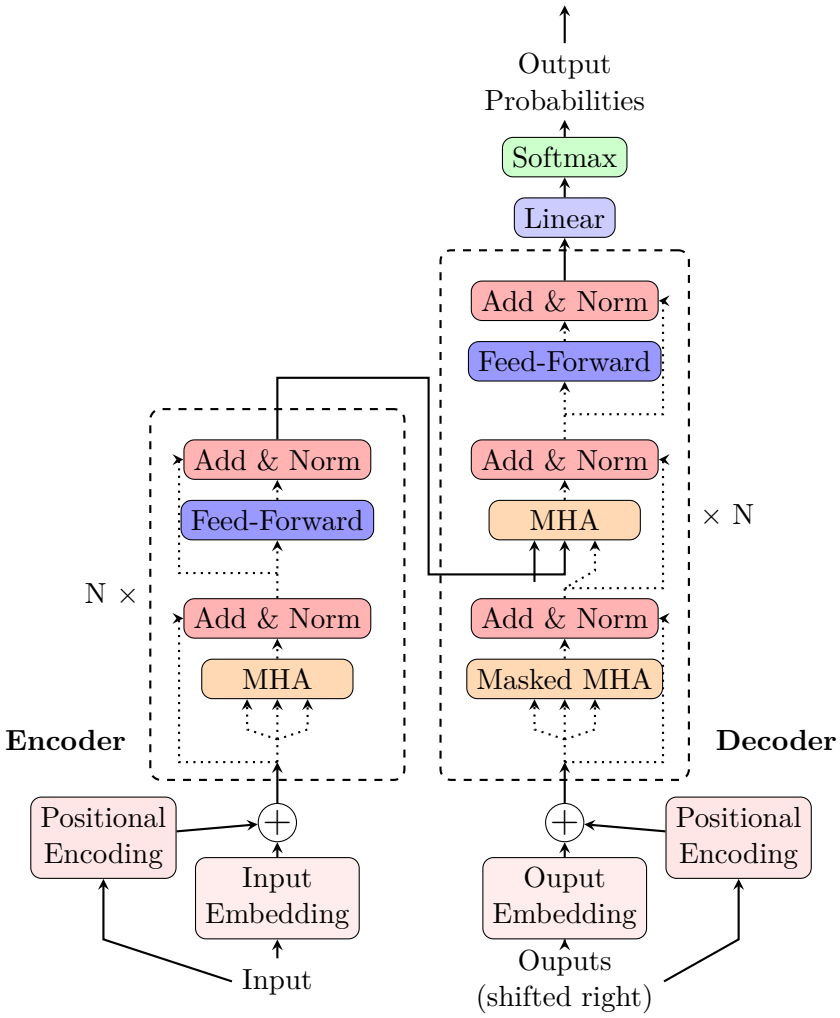


Figure 2.4: Transformers topology (inspired by [41]

Transformer architecture, as shown in figure 2.4, is composed of 3 main components : embedding (including positional encoding), encoder and decoder. The embedding consist of

representing the words in a vector space, using *tokenizer* to encode the sentence. 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.

The encoder is a stack of MHA with feed-forward layers, with the addition of a residual connection and a normalization layer between the two. The output of the encoder is used as the input of second MHA of the decoder. Like the encoder, the decoder is a stack of MHA with feed-forward layers, but is composed of two MHA. The first one is masked, to learn only the part before the word to predict, and the second one is not masked to learn the whole sentence.

This topology is the base of LLM, with diversification on the number of layer, the number of head and the size of the embedding vector. It will be further discussed in section 2.1.5, but some model are using only part of the transformer architecture, with the rise of *encoder-only* and *decoder-only* models.

2.1.4 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 2.1: 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. Figure 2.5 and table 2.1 illustrate the scope of the fine-tuning process, in opposition to the pre-training process.

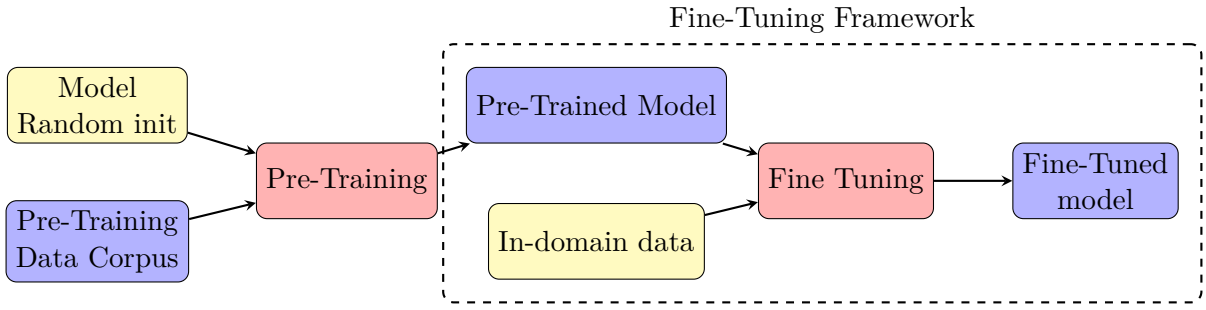


Figure 2.5: Pre-Training and Fine-Tuning Framework

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 [42], 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 2.1 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 methods are aiming to reduce the cost of fine-tuning, and make it more accessible to a wider range of users. Article [11] make an exhaustive review of PEFT, and is the base of this paragraph. The two main approaches to PEFT are *additive* and *reparameterization* methods.

The *additive* approach aims to add news weights or layers to the models, and train only theses weights. Popular method use *adapter* layers, adding layer between layers of the model, as shown in figure 2.6. One con of this approach is the rise of the inference time implied by the addition of these layers to the model.

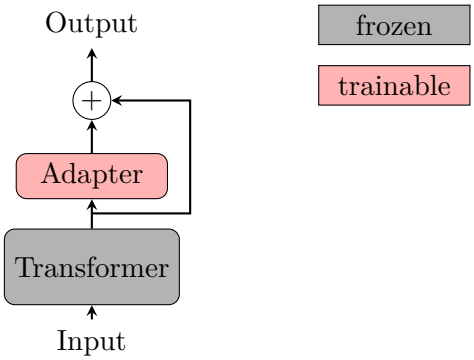


Figure 2.6: illustration of adapter layer

The *reparameterization* approach provide a proxy for model weights, to train this proxy and then merge it with the original model. The most popular method is Low Rank Adaptation (LoRA), based on article [15]. This method use the intrinsic rank of the weight matrix W, to factorize the matrix W into 2 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, as expressed in equation 2.4.

$$\begin{aligned} W = W_0 + \Delta W = W_0 + B.A \\ s.t. \quad W, W_0, \Delta W \in \mathbb{R}^{n*p}, \\ A \in \mathbb{R}^{r*p} \text{ and } B \in \mathbb{R}^{n*r} \end{aligned} \tag{2.4}$$

Figure 2.7 show the illustration of the Low Rank Adaptation (LoRA) layer, with A and B being the factorization of the weight matrix. 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.

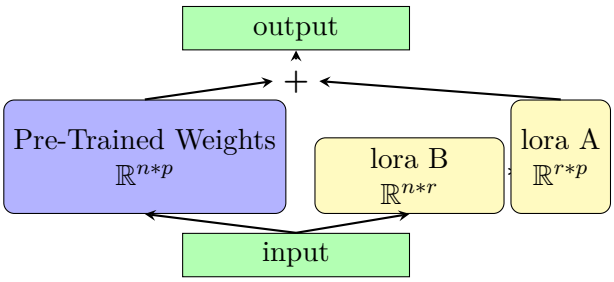


Figure 2.7: illustration of lora layer

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

2.1.5 Review and taxonomy of LLMs

Exhaustive review of LLMs can be found, like the article [33], 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[3] or its variations, 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) models, with it's web service ChatGPT [29] is a decoder-only model, and has strongly contribute to the renowned of the LLMs. We can also cite Llama [10] 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[23] or T5[32] 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[1] : 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.

2.2 Auto-DNN

Automated Deep Neural Networks (Auto-DNN), as defined in [38], 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 :

- Taking the human out of the loop, to find solutions outside of human expertise and way of thinking
- 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.

2.2.1 Problem Formulation

The general Auto-DNN optimization problem can be defined, like article [38], 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.

For reminder, namely (x, y) the input-output pair, $f(x, \omega) = \hat{y}$ the predicted output and $\mathcal{L}(\hat{y}, y)$ the loss function of the model. Subsequently to the optimization problem of section 2.1.1, ω^* is the optimal parameters of the model.

Following the first optimization problem, a second is expressed as the Auto-DNN problem, with equation 2.5. In this equation, f is the objective function, often the negative loss function or the accuracy.

$$a^* \in \arg\max_{a \in A} f(\Theta(a, d_{train}), d_{valid} = \arg\max_{a \in A} f(a). \tag{2.5}$$

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 [4] 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 2.2.4. The **performances estimation strategy** is the definition of the loss function as expressed in equation 2.1, and all linked attributes (e.g. datasets choice). This part can also includes approach like multi-fidelity, modifying the evaluation along iterations.

2.2.2 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* [4]. Figure 2.8 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 [21], which uses **weight sharing** methods, such as the approach proposed in [30].

The methods to solve NAS for LLM are quite diverse. Some methods encode topology into a continuous space, broadening the range of possible optimization techniques. Further details will be addressed in Section 2.2.4, but it is worth mentioning derivative-based methods like [24], which apply such techniques to NAS.

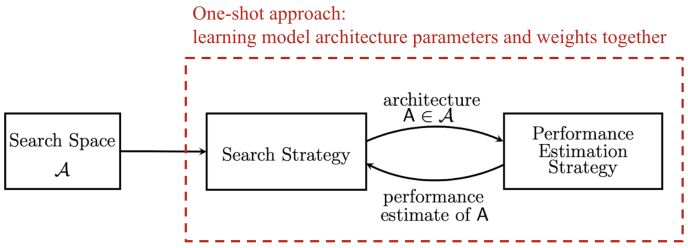


Figure 2.8: Neural Architecture Search Workflow

2.2.3 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 [5]. 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 automating the choice of the best hyperparameters for a specific problem (a quadruplet a as defined in 2.2.1). 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.

2.2.4 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. The search space $\mathcal{X}$ is generally a subspace of the real space $\mathbb{R}^n$, where n is the dimensionality of the problem.

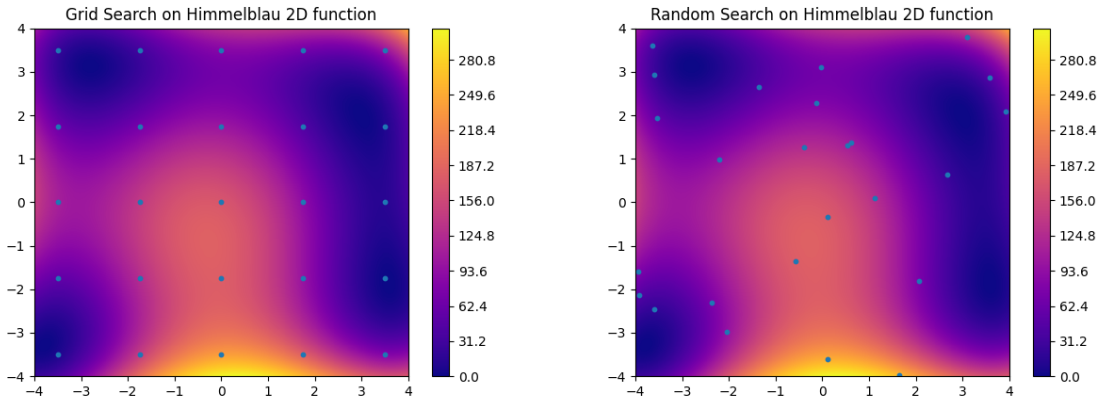
Optimization methods aim to efficiently navigate vast and complex search spaces, balancing the trade-off between *exploration* (searching new regions) and *exploitation* (refining known good solutions). In the context of hyperparameter optimization (HPO), global optimization plays a crucial role in systematically identifying configurations that maximize model performance while minimizing computational costs. The subsequent sections categorize and detail these optimization methods.

$$f(x) = (x_1^2 + x_2 - 11)^2 + (x_1 + x_2^2 - 7)^2 \tag{2.6}$$

In the next paragraphs, I will present a taxonomy of global optimization methods, w.r.t. their relevance in the HPO problem. Equation 2.6 represent the *himmelblau* function, a well-known non-convex function with multiple local optima. It will be used as an example in the following sections if relevant.

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.



(a) Grid Search (b) Random Search

Figure 2.9: Exploratory Methods on Himmelblau 2D function

Figure 2.9 show the results of the GS and RS on the *himmelblau* function, with a given budget of 25 evaluations. Con of GS is the curse of dimensionality, since with a grid of size s, the number of configurations to evaluate is d^n. RS is less dependent on the dimensionality since the number of configurations to evaluate is only fixed by the budget. With search space including a lot of good solutions in the space, RS can achieve efficient performance.

Metaheuristic Approaches

Metaheuristic methods, including Simulated Annealing (SA) and Evolutionnary Algorithms (EA), are designed to search more efficiently within large hyperparameter spaces by mimicking natural or physical processes. SA 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. EA, 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.

Metaheuristic can be classed on two main categories, population-based and solution-based. The first one like GA are methods working on a population of candidate solutions, while the second one like SA or Iterated Local Search (ILS) are methods working on a single solution. These approaches are particularly useful for complex, non-convex hyperparameter spaces where simpler methods like GS or RS might struggle.

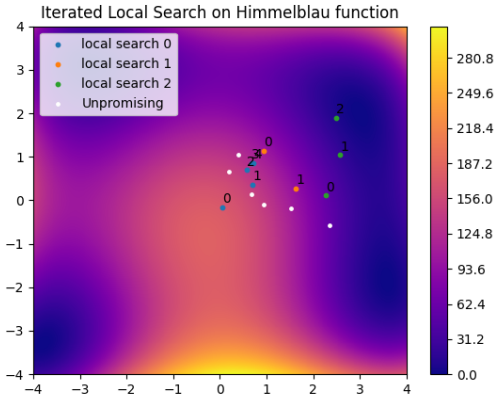


Figure 2.10: Example of Iterated Local Search on himmelblau

Even if theses methods are effective, they can be computationally expensive when dealing with expensive objective function evaluations, especially population-based methods. For a lot of methods like ILS in figure 2.10, unpromising evaluation are just discarded without really exploiting them. This characteristic makes them less suitable for high-dimensional hyperparameter spaces, and especially for HPO.

Partition Based Optimization (PBO)

Partition Based Optimization (PBO) methods aim to divide the hyperparameter search space into smaller subspaces, focusing the search on the most promising regions. This division can be done either by penalizing less promising regions or by favorizing. Famous PBO methods are Fractal Decomposition Algorithm (FDA)[27], DiRect [18] or even Simultaneous Optimistic Optimization (SOO)[26]. One of the biggest advantages of PBO methods is their intrinsicly parallelism abilities, enabling the scalibility of the optimization process when working with large hardware resources.

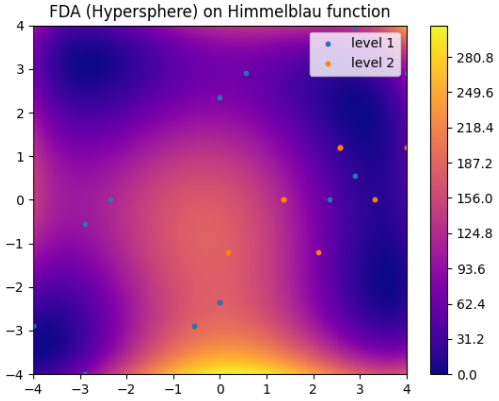


Figure 2.11: Example of a Partition Based Optimization on 2D himmelblau

Figure 2.11 shows an example of PBO algorithm, FDA using Hypersphere, on the *himmelblau* function. Article [6] provides an exhaustive review and taxonomy of PBO methods.

Surrogate-Model Based Optimization (SMBO)

Surrogate-Model Based Optimization (SMBO) approach is aimed to deal with expensive objective function, in terms of costs, times or limitations of experiments, originally used for mecanical engineering, or chimic experiments. In face of the costs of ML training, SMBO became the must have of Auto-DNN problem. Bayesian Optimization (BO) contains a lot of SMBO methods, using probabilistic models. Article [35] provides an exhaustive review and taxonomy of BO.

The global idea is to reduce the cost of the objective function by using surrogate models to predict the values of the objective function. The surrogate can be parametric like *Thompson Sampling in the Beta-Bernoulli Bandit Model*, or non-parametric like Gaussian Process (GP). The model is used to extract an *Acquisition function*, which is optimize to obtain best promising point to evaluate.

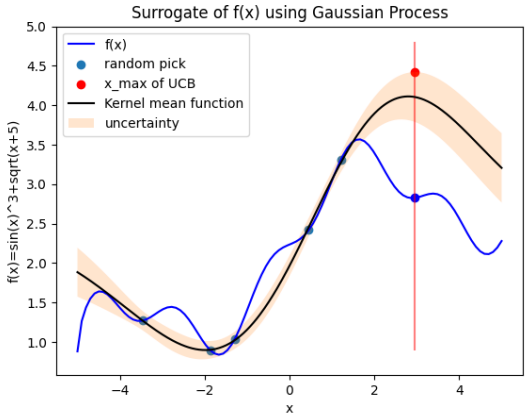


Figure 2.12: Gaussian Process example on $f(x) = \sin(x)^2 + \sqrt{x+5}$

Figure 2.12 shows an example of Gaussian Process (GP) on a simple function. Based on historic of points, the gaussian process is able to compute a mean and a variance, which is used to compute the acquisition function. The acquisition function is then used to select the next point to evaluate. More details will be discussed in section 3.4. The biggest con of SMBO methods are the sequential nature of the optimization process, which limits the scalability of the optimization process when working with large hardware resources.

2.2.5 Parallel Optimization and High Performance Computing

The advent of largest DNN was possible with the development of Graphics Processing Unit (GPU) and corresponding software. Parrellel computation and HPC are what makes the uses of complex parallel architectures possible, from multicore processors to distributed clusters and grids.

Software parallelism for optimization algorithm can be achieved in mutliples ways, from algorithmic-eval parallelism, to solution-level parallelism, explained more in details in [37]. In this internship, I will focus on solution-level parallelism in intra-nodes GPU clusters.

As said in chapter 1, my internship is funded on NumPEx project. NumPEx, through its Exa-MA (Methods and Algorithms for Exascale) subprogram, addresses computational challenges of exascale systems by developing scalable algorithms and architectures. This work aims to align with Exa-MA's goals by presenting Proof-of-Concept of a scalable optimization algorithm for LLM.

2.3 LLMs application to manufacturing context

To link this internship to my engineering curriculum, I will explore the application for LLM to manufacturing. The application can be split in three : Quality Control (QC), Supply Chain Management (SCM) and 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.

2.3.1 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" [43]. 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.

2.3.2 LLMs-based supply chain management

Since decades, **demand forecast** is a hot field of Supply Chain Management (SCM), with many methods considering means and data collection of the company. LLM, and in specific time-LLM [17], 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.

2.3.3 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.

2.4 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 LLM won't be the priority

After a first week, the possible fields was narrowed to two tracks. The first one is based on article [21], and involve the pruning of a large pre-trained model, in order to reduce latency without losing to much accuracy. The other one [40] is Hyper-Parameter Optimization 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 litterature 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 3
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 2.9b), every experiments of chapter 4 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 3.3 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.

3.1 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 2 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/

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 [41, 38], establishing the core of the domain, and reviews like article [4, 38], 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.

3.2 Blackbox Elaboration

My internship can be seen as global optimization applied to a noisy, mixed-variables, 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), without any information about the internal process.

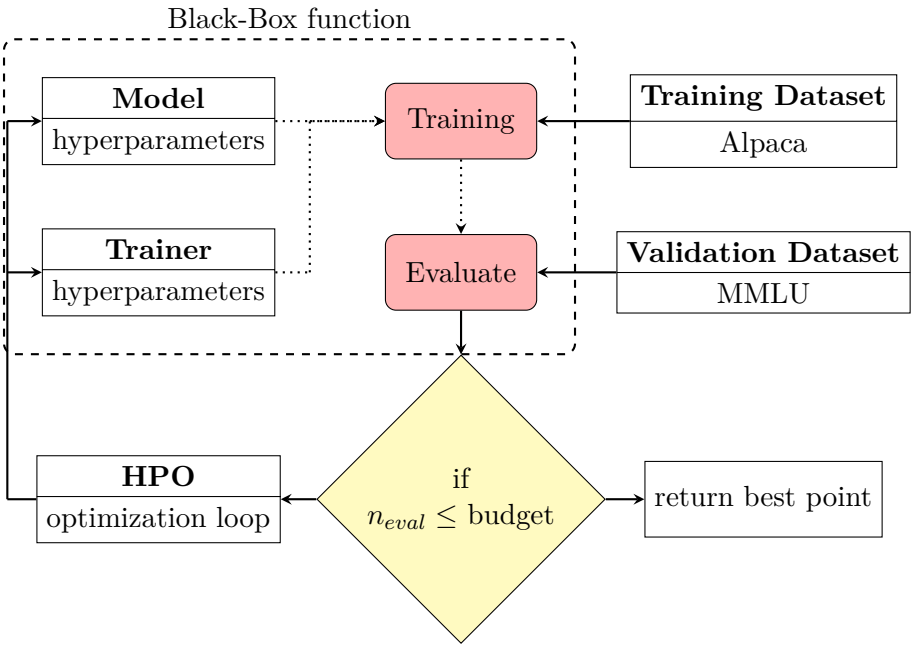


Figure 3.1: HPO workflow

The blackbox process here is described by figure 3.1. 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 3.1. The reproduction of the blackbox function using Python is done with a *ModelEvaluator* class, reproducing the nexts parts.

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

3.2.1 Fine-Tuning of the Model

For fine-tuning, the first step is to choose the model to work with. For this choice, the first element was the kind of tasks we want to work with. For the biggest use case and impact, the focus is done on *Decoder-only* model. Then, based on article [40], and open-source model availability, I choose to work with a model of LlaMa family.

The LlaMa family, launched on February 24, 2023 with the publication of "LLaMA: Open and Efficient Foundation Language Models"[39], is a family of open-source (topology and weights values) *decoder-only* fundational models produced and maintained by Meta AI. Latest releases from september 2024, LlaMa 3[10] set, include model from one billion of parameters (*LlaMa 3.2-1B*) to 405 billions of parameters (*LLaMA 3.1-405B*), and achieved State-of-the-art performances on benchmarks. During the first phase of the elaboration of the fine-tuning, I work with *TinyLlama-1.1B*, a lightweight model based on LlaMa architecture. After this phase, I upgraded to *LlaMa 3.2-3B* for a better fidelity compared to high performance models, but compatible with hardware constraints described in section 3.6.

After the model, the next step is the training dataset. The reference in fine-tuning training dataset is the *Alpaca* dataset[12]. It's an AI-generated dataset of 52k examples of instruction-based dialogues from the *Stanford Alpaca* project. The dataset is composed of 3 fields : *input*, *output*,*instruction* and *text*. At first, I used *Alpaca-2K* datasets, a small subset of *Alpaca* dataset composed of 2k examples. Then, I used the full *Alpaca* dataset when I reached a stable version.

For the training of the weights, as described in 2.1.1, I use AdamW, a variant of Adam decoupling weigth decay [22] from learning rate. Along with the optimizer, the training went with Low Rank Adaptation (LoRA) as a Parameter Efficient Fine-Tuning (PEFT) methods, as defined in section 2.1.4. The fine-tuning follow the generic ANN training process, except only LoRA are trainable. LoRA is applied to all weights inside Multi-Head Attention, i.e. keys, weights, queries and output weights, so the linear layers outside MHA are not affected.

For the implementation, at first I started from example from PyTorch Lightning documentation, then I adapted it to my needs. This approach used PyTorch as backend, providing *LightningModule* and *LightningDataModule* classes. GPT specific function and classes were implemented in litgpt librairy. For loading models, HuggingFace, the standard hub for model and datasets, is used to manage token with Meta interface. After few adaptations, I had python code almost usable for fine-tuning, but the file input and output at each step (after training, merging with LoRA weights, conversion for evaluation) was prone to error and file corruption.

In the last half of December, I decided to restart this part from scratch, using solely litgpt library with it's Command Line Interface (CLI). This approach was easier to implement, and provided a more stable workflow although it reduced the training performance, using another parallel strategy. In this approach, I managed long string corresponding to CLI commands, and I used python *subprocess* to execute them.

Figure 3.1 summarize the fine-tuning process, to understand global process of this part. The process in taking model and hyperparameters as input, to load model and dataset from HuggingFace librairy. Then LoRA is implemented according to hyperparameters, using litgpt librairy. After that, the model is trained using PyTorch Lightning, with the classical ANN training process.

Algorithm 3.1: Fine-Tuning(model, hyperparameters)

Input: model, hyperparameters
model ← load ("LlaMa") ▷load PyTorch Lightning model using HuggingFace lib
model ← lora(model, hyperparameters) ▷apply LoRA to model using litgpt lib
$x, y \leftarrow$ load("Alpaca") ▷load dataset using HuggingFace lib
▷automated by PyTorch Lightning
foreach $(x_i, y_i) \in (x, y)$ **do**
  $\hat{y} \leftarrow$ model.forward(x) ▷forward pass
  loss $\leftarrow \mathcal{L}(\hat{y}, y)$ ▷compute loss
  model ← backpropagation(loss) ▷backward pass
end
return model

3.2.2 Evaluation of the model

To evaluate an ANN, the standard way is to split the dataset into training and validation datasets. The training dataset is used to train the model, and the validation dataset is used to evaluate the model. The evaluation metric can be the loss, a metric about the difference between the predicted output and the true output, or the accuracy, a metric about the percentage of correct predictions. There exists differents kind of loss, link cross-entropy, or mean-square error, to adapt to the datasets and the problem.

With LLM, the diversity of the tasks, even with a *decoder*-only model, is crucial. During the training, the loss or the accuracy is done with the prediction of the next word, compared to the true one. It does not represent the generalization capability of the model. To deal with it, challenge benchmarks, often using Multi-Choice Question (MCQ) on diverses thematics, were rising. It's was enhanced by article like [42], proving the advantage of fine-tuning in terms of generalization.

Among those challenge benchmark datasets, I choose two of them : one to use during HPO and the other to look at overfitting. The Hellaswag [46] dataset is composed of 40k lines of text and 4 choice of answers, meaning random pick lead to 25% of accuracy. I use this first during HPO. The MMLU dataset [13] is a dataset over multiples subjects, use to prevent overfitting.

The implementation of this part is done with litgpt library, as a CLI like for training. Under the litgpt part, it's using lm_eval library from HuggingFace to manage the evaluation of the accuracy.

3.3 Search Space Definition

The search space is defined with the choice of hyperparameters, theirs bounds, theirs types and even the scale of theirs steps. Well-defined search space is crucial for a correct application of HPO : if the space is too high-dimensional, the HPO algorithm will need too many shots to converge to a good solution, if it's too small, we are missing it's relevance.

The search space is composed of 5 hyperparameters, from classical training hyperparameters to LoRA specific ones. A detailed presentation of hyperparameters is just below, but one can look at table 3.1 for a summary.

- LoRA rank : with LoRA methods, the fine-tuning weights matrix $\Delta W \in \mathbb{R}^{n*p}$ is replaced by two matrices $A \in \mathbb{R}^{r*p}$ and $B \in \mathbb{R}^{n*r}$ with $\Delta W = B * A$. r is called the LoRA rank, and scale the reduction of the weights matrix. It's an integer, and it's value range from 1 to 512, following article [**tribes**] indication.
- LoRA scale (α) : when merging pre-trained weights W_0 and Lora fine-tuned weights $B * A$, the scale α is weighting the influence of fine-tuning, with $W = W_0 + \frac{\alpha}{r} * (B * A)$. It's a continuous value, from 1 to 100.
- Learning rate : the learning rate is a classical hyperparameter used in HPO, weighting the gradient of each weight when doing backpropagation. It is often tuned in a logarithmic scale, to manage effectively the exploration.
- Dropout probability : based on article [36], dropout is a method used to prevent over-fitting, by randomly fixing cells/layers to zeroes during one iteration. Being a probability, it's bounds by 0 and 1.
- Weight decay : weight decay is used to improve generalization capacity, as proved in article [22], by reducing the weights at each iterations by a small value, to force the model to use new inputs. Typically, the parameter for weight decay is set on a logarithmic scale between 10^{-3} and 10^{-1}.

Hyper-parameter	**Optimization range**		**Conversion**
	Lower Bound	**Upper Bound**	
Learning Rate	-10	-1	$f(x) = 10^x$
LoRA Rank	2	32	$f(x) = \text{round}(x)$
LoRA scale (α)	16	64	$f(x) = \text{round}(x)$
LoRA Dropout	0	0.5	$f(x) = x$
Weight Decay	-3	-1	$f(x) = 10^x$

Table 3.1: Summary of Hyperparameter Search Space

For the 2 integers variables (LoRA rank and LoRA scale), to adapt to continuous optimization algorithms, the relax and round methods will be applied. It mean that the integers constraints is relaxed when generating a solution, and is rounded when evaluating a solution. Others methods like computing with lower and upper discrete value can be used, but this one was kept for simplicity and computation costs. For the 2 variables with logarithmic scale (learning rate and weight decay), to explore with consistency the search space, the optimization algorithm will be bound between the exponential values of the bounds, and a logarithm will be applied when evaluating a solution.

3.4 Optimization Algorithms

Linked with section 2.2.4, this part aims to describe the implemented algorithms, and how they are applied to the optimization problem. It start from elements of section 2.2.4, then describe algorithms and show examples of application.

The first approach to explore is Surrogate-Model Based Optimization (SMBO), and particularly Bayesian Optimization (BO) using Gaussian Process (GP) (BO-GP). Then considering the dimensionnality of the problem, and the PBO performance benchmark in article [7], I went with Simultaneous Optimistic Optimization (SOO) algorithm as representative of PBO methods. After theses two approachs, section 3.4.3 present an hybrid approach, combining the intrinsic parallel abilities of PBO combined to the efficiency and exploitation of BO.

3.4.1 Bayesian Optimization (BO)

Bayesian Optimization is often defined as a "sequential model-based approach to solving problem" [35]. A surrogate model is used to build a posterior considering a prior knowledge formed on known points. On this posterior, an acquisition function is compute to act as the surrogate function for the function to optimize. On this work, a focus is done on GP for the BO surrogate. GP use the kernel trick to build a bayesian nonparametric regression model. It use a mean vector m_i and a covariance matrix $K_{i,j}$ to define the prior function :

$$\mathrm{f}|X \sim \mathcal{N}(m, K) \tag{3.1}$$

From the prior function and the data points $\mathcal{D}$, the GP build a posterior. On this posterior is build an acquisition function used as a surrogate for the objective function.

Algorithm 3.2: BO

Input: Ω, f, K_D, $\mathcal{O}$, f_{acq}, n_{init}, n_{opt}

// initiate function

for $i \leftarrow 1$ **to** n_{init} **do**
  $\lambda' \leftarrow \text{LHS}(\Omega, \mathcal{D})$ // Sample one point
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\lambda', f(\lambda'))\}$ // Add solution and evaluation to set of data
end
for $i \leftarrow 1$ **to** n_{opt} **do**
  $\mu_D, K_D \leftarrow \text{Update}(K_D, \mathcal{D})$
  $K_D \leftarrow \text{Fit}(\text{GP}(K_D), \mathcal{D})$
  $\lambda' \leftarrow \text{Optimize}(f_{\text{acq}}(K_D), \mathcal{O})$ // Generate new point
  $\mathcal{D} \leftarrow \mathcal{D} \cup \{(\lambda', f(\lambda'))\}$ // scoring function
end
return best of $\{(\lambda^*, f(\lambda^*)) \in \mathcal{D}\}$

Algorithm 3.2 offer an overview of the BO process. To ease the first build of the surrogate, it's crucial, as proven in article [44], to sample efficiently the search space. This sampling provides information for the Gaussian Process to estimation the function. Like article [2], Latin Hypercube Sampling (LHS) is used as a sampling method, for a defined budget called n_init.

After this preliminary phase, a second phase is done with on loop containing the update of the Gaussian Process, the optimization of the acquisition function to obtain a new points to evaluate. After the evaluation of the point, the point is added to the history $\mathcal{D}$ and so on. The loop end based on a budget n_{opt}.

For this algorithm, the first requirements is the search space, and the objective function already described in 3.3 and **??** respectively. On the GP part, we need to define a Kernel function $K_{\mathcal{D}}$, an acquisition function f_{acq} and an Inner Optimizer $\mathcal{O}$. The acquisition function is logEI, more reliable than EI, based on article [**ament_unexpected_2024**]. The kernel and the inner optimizer are the standard implementation of Botorch, introduced in the next paragraph, with a radial basis function kernel and multi-start optimization method.

BoTorch [**balandat_botorch_2020**] is a Bayesian Optimization library built on PyTorch, designed for efficient and scalable optimization of expensive black-box functions. Leveraging PyTorch's GPU acceleration and integration with GPyTorch [**gardner_gpytorch_2021**] for Gaussian Processes, BoTorch enables advanced surrogate modeling and optimization. Botorch

is used on this work for all tasks including GP, this part and section 3.4.3

3.4.2 Simultaneous Optimistic Optimization (SOO)

In global optimization, a lot of methods are based on the partition of the search space [27, 18, 26]. Theses approaches are mostly deterministic, and enhance intrinsic parallelization ability. For theses methods, the dimensionality of the problem is a key to choose the specific algorithm. With a dimensionality around 5, based on benchmarks at the end of article [7], the Simultaneous Optimistic Optimization (SOO) [26] algorithm seems a good way to start.

Algorithm 3.3: SOO

Input: Ω, f, K, $n_{\max}$
// initiate
$x_{0,0} \leftarrow \text{center}(\Omega)$
$f_{0,0} \leftarrow f(x_{0,0})$
$\mathcal{T}_1 \leftarrow \{x_{0,0}, f_{0,0}, \Omega\}$
$n \leftarrow 1$
while $n < n_{max}$ **do**
  $\nu_{\max} \leftarrow -\infty$
  for $h \leftarrow 0$ **to** $depth(\mathcal{T}_n)$ **do**
    $j \leftarrow \arg\max_{j \in \{j | (h,j) \in L_n\}} f(x_{h,j})$ // select function
    if $f(x_{h,j}) > \nu_{max}$ **then**
      $\Omega_{h+1,j+1}, \ldots, \Omega_{h+1,j+K} \leftarrow \text{section}(\Omega_{h,j}, K)$
      for $i \leftarrow 1$ **to** K **do**
        $n \leftarrow n + 1$
        $x_{h+1,j+i} \leftarrow \text{center}(\Omega_n)$
        $f_{h+1,j+i} \leftarrow f(x_{h+1,j+i})$ // Scoring function
        $\mathcal{T}_n \leftarrow \{(x_{h+1,j+i}, f_{h+1,j+i}, \Omega_{n+1})\}$ // add_leaf function
        $\nu_{\max} \leftarrow f_{h,j}$
      end
    end
  end
end
return best of $x_{h,j}, f(x_{h,j})$

SOO is a tree-based space partitioning method for black-box optimization, inspired by Monte Carlo Tree Search (MCTS) methods. SOO is called optimistic since it assume the existence of l such that $f(x^*) - f(x) \leq l(x, x^*)$ where x^* is the maximizer of x.The algorithm partition the space Ω by building a tree with smaller and smaller sub-space Ω_n. A node (h, j), the node number j of depth h, is scored at the center of his space.

An expanded node have K children, making the tree a K-nary tree. L_n is the *open list* of the tree, to avoid expanding the same nodes over and over. At each round, SOO expand a maximum of one node by depth, meaning that each round score a maximum of $depth * (K)$ solution, enhancing the parallel evaluation of the solution. Summary of SOO is present in algorithm 3.3

The original algorithm manage the end of the loop with the $h_{max}(n)$ function, limiting the depth of the tree search. To compare different algorithm, the stopping criterion here is n_{max},

the evaluation budget.

3.4.3 Bayesian Multi Scale Optimistic Optimization (BaMSOO)

Surrogate-Model Based Optimization (SMBO) algorithms harness the exploitation of the informations to define a cost-reduce function to optimize. This approach ensure exploitation but have several limitations, including the parallelization difficulties [**X**, **X**]. On the other hand, Partition-based approach are massively parallel, but are computation costly in front of very expensive objective function. To overcome both limitations, hybrid methods, using surrogates and space partition, were developed.

In this work, we focus on BaMSOO, a SOO based algorithm (algorithm A.1). Like SOO, BaMSOO performs a K-inary partitionning of the space, using the center of the partition to evaluate.

$$\begin{aligned} \mathcal{UCB}(x|\mathcal{D}_t) &= \mu(x|\mathcal{D}_t) + B_N * \sigma(x|\mathcal{D}_t) \\ \text{with } B_N &= \sqrt{2\log(\pi^2 N^2/6\eta)}, \eta \in (0,1) \end{aligned} \tag{3.2}$$

The difference lies primarily in the scoring $g(.)$ of the partitions (lines 25 to 30). In the face of an expensive objective function, BaMSOO leverages a GP surrogate to estimate the potential of a point, using the $\mathcal{UCB}$ as a measure of expected performance. Given a partition with center x and existing evaluations $\mathcal{D}_t$, the $\mathcal{UCB}$ of x, defined in Equation 3.2, is compared against the best evaluation so far, f^+. If the $\mathcal{UCB}$ is higher than f^+, the algorithm evaluates x directly using the objective function $f(.)$. Otherwise, the partition is scored using the LCB of x, reflecting the lower bound of potential improvement.

Algorithm 3.4: BamSOO scoring

```
1  if UCB(x_{h+1,j+i}, μ, σ) ≥ f⁺ then
2  |   g_{h+1,j+i} ← f(x_{h+1,j+i})
3  |   t ← t + 1
4  end
5  else
6  |   g_{h+1,j+i} ← LCB(x_{h+1,j+i}, μ, σ)
7  end
8  if g_{h+1,j+i} > f⁺ then
9  |   f⁺ ← g_{h+1,j+i}
10 end
11 n ← n + 1
12 T_n ← {(x_{h+1,j+i}, f_{h+1,j+i}, Ω_{h+1,j+i})}
13 return best of x_{h,j}, g(x_{h,j})
```

To sum up, this algorithm efficiently navigates the search space by leveraging partition-based exploration and utilizing the gathered information to prioritize regions with higher potential. By balancing exploration of untested partitions and exploitation of known promising areas, BaMSOO optimally allocates the evaluation budget, ensuring that computational resources are directed toward regions most likely to contain optimal solutions. This hybrid approach significantly mitigates the limitations of traditional methods, combining the global scalability of

partitioning with the precision of GP-based surrogate modeling.

For the implementation of the GP components, including the calculation of LCB and $\mathcal{UCB}$ scores, the BoTorch library was employed. This choice ensures computational efficiency and robustness, as BoTorch provides a modular framework for Bayesian optimization and GP modeling, seamlessly integrating with the partition-based structure of BamSOO. By adhering to the methodology outlined in section 3.4.1, the framework ensures consistency in surrogate modeling and acquisition function computation, further enhancing the effectiveness of the algorithm in high-dimensional, continuous search spaces.

3.5 Concrete Implementation

To implement what's described in section 3.2 to 3.4, I used Python as my main language. After a first phase of coding only with function on a small number of file, I started to rethink everything as Object Oriented Programming (OOP). When the rework from scratch happened in December, I used this opportunity to structure my code with OOP.

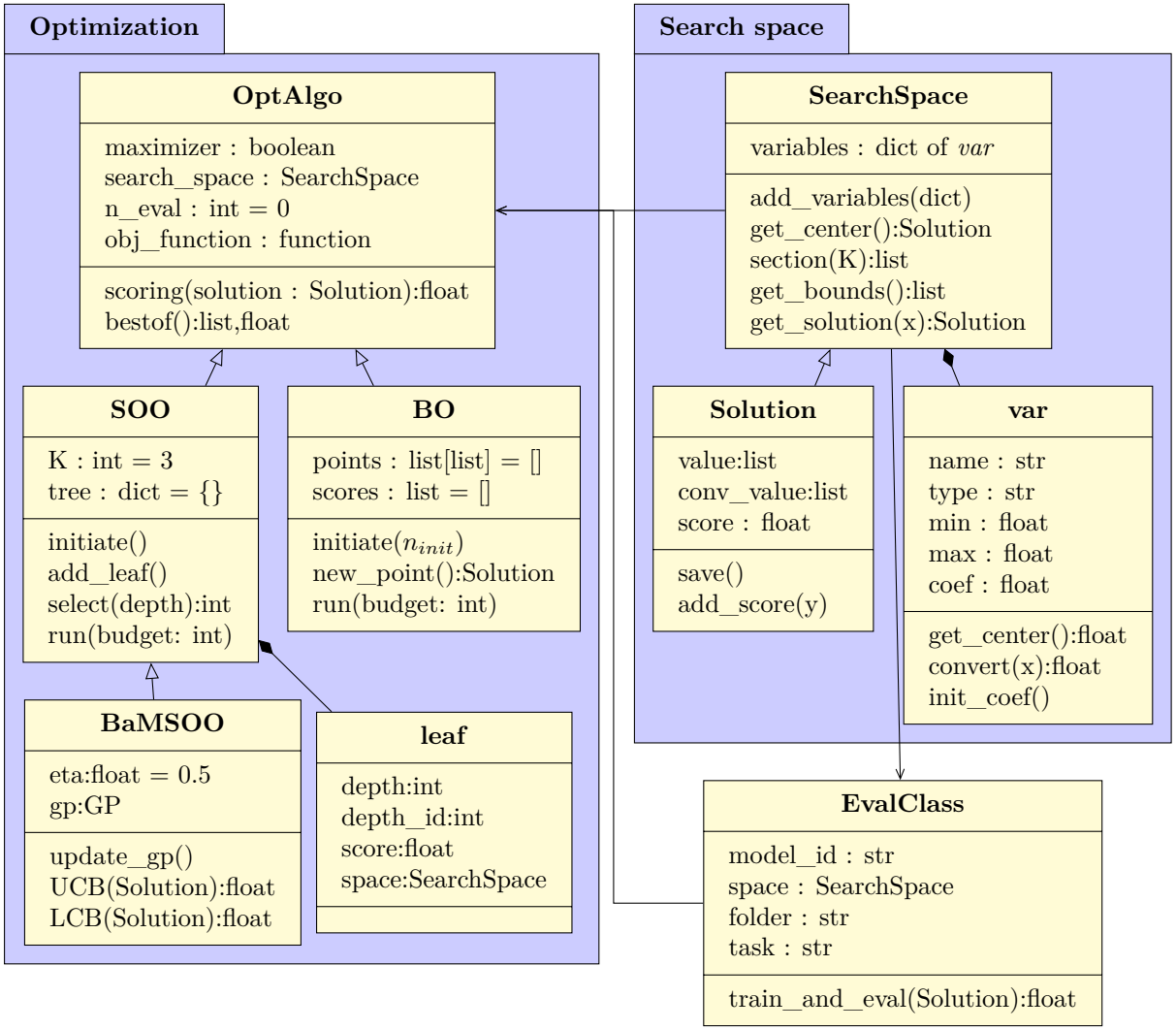


Figure 3.2: Class diagramm of the optimization framework

Figure 3.2 is an UML class diagramm presenting the whole framework of my internship, split in tree parts : the optimization part, including all optimization algorithm seen in section 3.4, the search space part, and evaluation part/

At the left, the optimization part includes a base class for optimization algorithm, managing all recurrent tasks, especially scoring and extracting the best result. From this class is built SOO and BO classes. All theirs attributes and functions are described in section 3.4.2 and 3.4.1 respectively, with comment in algorithms to precise which function is used for which line. SOO being a tree-based algorithm, a leaf class in create to manage the leaf of the tree, and especially the decomposition of the space. From SOO class, BaMSOO is built, add especially the GP surrogate, and updating the scoring function with $\mathcal{UCB}$ and LCB to adapt to algorithm 3.4.

At the top right, the search space part is firstly composed with a search space class. This class is composed on multiple var class, use to automate all function used by each variables. Then, it's used to manage the space, like section to split the space for SOO, or other functions to deal with algorithms and compatibility needs. Then, the Solution class inherit the search space class, to force a solution to be define in a search space, and then manage the conversion of the solution, with the conversion function of table 3.1.

The last part is the eval class, use to store the model id, the experiment folder, the task and then call the *train_and_eval* function when the scoring function is called. The whole structure allow an easy understanding of the framework, an easier testing and debuging process and a better reusability of each part of the code.

Along these part, all experiments are also stored as a subpackage of the whole, and then called from the main file. This method ease the reproducibility of the experiments, and avoid import or module error from python package management.

3.6 experiments setup

Grid5000, chuc etc.

3.7 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 4
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.

4.1 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.

4.2 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.

4.3 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 5
Reflexion and Prospection

5.1 Generalisation of the work

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

5.2 an industrial context

5.3 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] Yejin Bang et al. *A Multitask, Multilingual, Multimodal Evaluation of ChatGPT on Reasoning, Hallucination, and Interactivity*. arXiv:2302.04023 [cs]. Nov. 2023. DOI: 10.48550/arXiv.2302.04023. URL: http://arxiv.org/abs/2302.04023 (visited on 12/17/2024).

[2] Prapatsorn Borisut and Aroonsri Nuchitprasittichai. "Adaptive Latin Hypercube Sampling for a Surrogate-Based Optimization with Artificial Neural Network". en. In: *Processes* 11.11 (Nov. 2023). Number: 11 Publisher: Multidisciplinary Digital Publishing Institute, p. 3232. ISSN: 2227-9717. DOI: 10.3390/pr11113232. URL: https://www.mdpi.com/2227-9717/11/11/3232 (visited on 01/07/2025).

[3] 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).

[4] 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).

[5] Matthias Feurer and Frank Hutter. "Hyperparameter Optimization". en. In: *Automated Machine Learning: Methods, Systems, Challenges*. Ed. by Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. Cham: Springer International Publishing, 2019, pp. 3–33. ISBN: 978-3-030-05318-5. DOI: 10.1007/978-3-030-05318-5_1. URL: https://doi.org/10.1007/978-3-030-05318-5_1 (visited on 12/13/2024).

[6] T. Firmin and E-G. Talbi. "A Comparative Study of Fractal-Based Decomposition Optimization". en. In: *Optimization and Learning*. Ed. by Bernabé Dorronsoro et al. Cham: Springer Nature Switzerland, 2023, pp. 3–20. ISBN: 978-3-031-34020-8. DOI: 10.1007/978-3-031-34020-8_1.

[7] Thomas Firmin and El-Ghazali Talbi. "A fractal-based decomposition framework for continuous optimization". July 2022. URL: https://hal.science/hal-04474444 (visited on 01/07/2025).

[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] Yoav Goldberg. "A Primer on Neural Network Models for Natural Language Processing". en. In: *Journal of Artificial Intelligence Research* 57 (Nov. 2016), pp. 345–420. ISSN: 1076-9757. DOI: 10.1613/jair.4992. URL: https://www.jair.org/index.php/jair/article/view/11030 (visited on 12/17/2024).

[10] Aaron Grattafiori et al. *The Llama 3 Herd of Models*. arXiv:2407.21783 [cs]. Nov. 2024. DOI: 10.48550/arXiv.2407.21783. URL: http://arxiv.org/abs/2407.21783 (visited on 12/13/2024).

[11] Zeyu Han et al. *Parameter-Efficient Fine-Tuning for Large Models: A Comprehensive Survey*. arXiv:2403.14608 [cs]. Sept. 2024. DOI: 10.48550/arXiv.2403.14608. URL: http://arxiv.org/abs/2403.14608 (visited on 12/13/2024).

[12] Rohan Taori and Ishaan Gulrajani and Tianyi Zhang and Yann Dubois and Xuechen Li and Carlos Guestrin and Percy Liang and Tatsunori B. Hashimoto. *Stanford Alpaca: An Instruction-following LLaMA model*. publisher : GitHub. Dec. 2024. URL: https://github.com/tatsu-lab/stanford_alpaca (visited on 12/18/2024).

[13] Dan Hendrycks et al. *Measuring Massive Multitask Language Understanding*. arXiv:2009.03300 [cs]. Jan. 2021. DOI: 10.48550/arXiv.2009.03300. URL: http://arxiv.org/abs/2009.03300 (visited on 12/05/2024).

[14] Timothy Hospedales et al. "Meta-Learning in Neural Networks: A Survey". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 44.9 (Sept. 2022). Conference Name: IEEE Transactions on Pattern Analysis and Machine Intelligence, pp. 5149–5169. ISSN: 1939-3539. DOI: 10.1109/TPAMI.2021.3079209. URL: https://ieeexplore.ieee.org/abstract/document/9428530 (visited on 12/17/2024).

[15] 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).

[16] *Inria Joint Center*. en-US. URL: https://www.microsoft.com/en-us/research/collaboration/inria-joint-centre/ (visited on 11/25/2024).

[17] Ming Jin et al. *Time-LLM: Time Series Forecasting by Reprogramming Large Language Models*. arXiv:2310.01728 [cs]. Jan. 2024. DOI: 10.48550/arXiv.2310.01728. URL: http://arxiv.org/abs/2310.01728 (visited on 12/18/2024).

[18] D. R. Jones, C. D. Perttunen, and B. E. Stuckman. "Lipschitzian optimization without the Lipschitz constant". en. In: *Journal of Optimization Theory and Applications* 79.1 (Oct. 1993), pp. 157–181. ISSN: 1573-2878. DOI: 10.1007/BF00941892. URL: https://doi.org/10.1007/BF00941892 (visited on 12/18/2024).

[19] Abbas Khosravi et al. "Comprehensive Review of Neural Network-Based Prediction Intervals and New Advances". In: *IEEE Transactions on Neural Networks* 22.9 (Sept. 2011). Conference Name: IEEE Transactions on Neural Networks, pp. 1341–1356. ISSN: 1941-0093. DOI: 10.1109/TNN.2011.2162110. URL: https://ieeexplore.ieee.org/abstract/document/5966350 (visited on 12/17/2024).

[20] 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).

[21] 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).

[22] Anders Krogh and John Hertz. "A Simple Weight Decay Can Improve Generalization". In: *Advances in Neural Information Processing Systems*. Vol. 4. Morgan-Kaufmann, 1991. URL: https://papers.nips.cc/paper_files/paper/1991/hash/8eefcfdf5990e441f0fb6f3fad709e21-Abstract.html (visited on 01/07/2025).

[23] Mike Lewis et al. "BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension". en. In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, 2020, pp. 7871–7880. DOI: 10.18653/v1/2020.acl-main.703. URL: https://www.aclweb.org/anthology/2020.acl-main.703 (visited on 12/17/2024).

[24] 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).

[25] Warren S. McCulloch and Walter Pitts. "A logical calculus of the ideas immanent in nervous activity". en. In: *The bulletin of mathematical biophysics* 5.4 (Dec. 1943), pp. 115–133. ISSN: 1522-9602. DOI: 10.1007/BF02478259. URL: https://doi.org/10.1007/BF02478259 (visited on 11/20/2024).

[26] Rémi Munos. "Optimistic Optimization of a Deterministic Function without the Knowledge of its Smoothness". In: *Advances in Neural Information Processing Systems*. Vol. 24. Curran Associates, Inc., 2011. URL: https://proceedings.neurips.cc/paper/2011/hash/7e889fb76e0e07c11733550f2a6c7a5a-Abstract.html (visited on 12/18/2024).

[27] A. Nakib et al. "Deterministic metaheuristic based on fractal decomposition for large-scale optimization". In: *Applied Soft Computing* 61 (Dec. 2017), pp. 468–485. ISSN: 1568-4946. DOI: 10.1016/j.asoc.2017.07.042. URL: https://www.sciencedirect.com/science/article/pii/S1568494617304623 (visited on 11/20/2024).

[28] *Numérique pour l'Exascale (NumPEx)*. fr. URL: https://anr.fr/fr/france-2030/programmes-et-equipements-prioritaires-de-recherche-pepr/numerique-pour-lexascale-numpex/ (visited on 11/25/2024).

[29] OpenAI et al. *GPT-4 Technical Report*. arXiv:2303.08774 [cs]. Mar. 2024. DOI: 10.48550/arXiv.2303.08774. URL: http://arxiv.org/abs/2303.08774 (visited on 12/13/2024).

[30] 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).

[31] *Plan Calcul*. fr. Page Version ID: 218696510. Sept. 2024. URL: https://fr.wikipedia.org/w/index.php?title=Plan_Calcul&oldid=218696510 (visited on 11/25/2024).

[32] Colin Raffel et al. "Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer". In: *Journal of Machine Learning Research* 21.140 (2020), pp. 1–67. ISSN: 1533-7928. URL: http://jmlr.org/papers/v21/20-074.html (visited on 12/17/2024).

[33] Mohaimenul Azam Khan Raiaan et al. "A Review on Large Language Models: Architectures, Applications, Taxonomies, Open Issues and Challenges". In: *IEEE Access* 12 (2024), pp. 26839–26874. ISSN: 2169-3536. DOI: 10.1109/ACCESS.2024.3365742. URL: https://ieeexplore.ieee.org/document/10433480/ (visited on 12/17/2024).

[34] Waseem Rawat and Zenghui Wang. "Deep Convolutional Neural Networks for Image Classification: A Comprehensive Review". In: *Neural Computation* 29.9 (Sept. 2017). Conference Name: Neural Computation, pp. 2352–2449. ISSN: 0899-7667. DOI: 10.1162/neco_a_00990. URL: https://ieeexplore.ieee.org/abstract/document/8016501 (visited on 12/17/2024).

[35] Bobak Shahriari et al. "Taking the Human Out of the Loop: A Review of Bayesian Optimization". In: *Proceedings of the IEEE* 104.1 (Jan. 2016). Conference Name: Proceedings of the IEEE, pp. 148–175. ISSN: 1558-2256. DOI: 10.1109/JPROC.2015.2494218. URL: https://ieeexplore.ieee.org/abstract/document/7352306/authors#authors (visited on 12/13/2024).

[36] Nitish Srivastava et al. “Dropout: A Simple Way to Prevent Neural Networks from Overfitting”. en. In: (2014).

[37] El-Ghazali Talbi. “Parallel Evolutionary Combinatorial Optimization”. en. In: *Springer Handbook of Computational Intelligence*. Ed. by Janusz Kacprzyk and Witold Pedrycz. Berlin, Heidelberg: Springer Berlin Heidelberg, 2015, pp. 1107–1125. ISBN: 978-3-662-43504-5 978-3-662-43505-2. DOI: 10.1007/978-3-662-43505-2_55. URL: http://link.springer.com/10.1007/978-3-662-43505-2_55 (visited on 11/20/2024).

[38] 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).

[39] Hugo Touvron et al. *LLaMA: Open and Efficient Foundation Language Models*. arXiv:2302.13971 [cs]. Feb. 2023. DOI: 10.48550/arXiv.2302.13971. URL: http://arxiv.org/abs/2302.13971 (visited on 12/18/2024).

[40] 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).

[41] 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).

[42] 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).

[43] *What is quality control (QC)?* en. URL: https://www.techtarget.com/whatis/definition/quality-control-QC (visited on 12/18/2024).

[44] James Wilson et al. “Efficiently sampling functions from Gaussian process posteriors”. en. In: *Proceedings of the 37th International Conference on Machine Learning*. ISSN: 2640-3498. PMLR, Nov. 2020, pp. 10292–10302. URL: https://proceedings.mlr.press/v119/wilson20a.html (visited on 01/07/2025).

[45] Yonghui Wu et al. *Google’s Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. arXiv:1609.08144. Oct. 2016. DOI: 10.48550/arXiv.1609.08144. URL: http://arxiv.org/abs/1609.08144 (visited on 11/20/2024).

[46] Rowan Zellers et al. *HellaSwag: Can a Machine Really Finish Your Sentence?* arXiv:1905.07830 [cs]. May 2019. DOI: 10.48550/arXiv.1905.07830. URL: http://arxiv.org/abs/1905.07830 (visited on 12/05/2024).

[47] G.P. Zhang. “Neural networks for classification: a survey”. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 30.4 (Nov. 2000). Conference Name: IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), pp. 451–462. ISSN: 1558-2442. DOI: 10.1109/5326.897072. URL: https://ieeexplore.ieee.org/abstract/document/897072 (visited on 12/17/2024).

[48] Zhilu Zhang and Mert Sabuncu. “Generalized Cross Entropy Loss for Training Deep Neural Networks with Noisy Labels”. In: *Advances in Neural Information Processing Systems*. Vol. 31. Curran Associates, Inc., 2018. URL: https://proceedings.neurips.cc/paper/2018/hash/f2925f97bc13ad2852a7a551802feea0-Abstract.html (visited on 12/17/2024).

⋆ ⋆ ⋆

Chapter A
Annexes

A.1 Algorithm

Algorithm A.1: BamSOO

Input: Ω,f,K,$n_{\max}$,K_D:

$x_{0,0} \leftarrow \text{center}(\Omega)$
$g_{0,0} \leftarrow f(x_{0,0})$
$\mathcal{T}_1 \leftarrow \{x_{0,0}, g_{0,0}, \Omega\}$ // Initiate the tree
$f^+ \leftarrow g_{0,0}$
$n \leftarrow 1$,$t \leftarrow 1$ // nodes and evaluation index
$\mathcal{D}_1 \leftarrow \{x_{0,0}, g_{0,0}\}$ // list of evaluated points
while $t < n_{max}$ **do**
  $\nu_{\max} \leftarrow -\infty$
  for $h \leftarrow 0$ **to** $depth(\mathcal{T}_n)$ **do**
    $j \leftarrow \arg\max_{j \in \{j|(h,j)\in L_n\}} g_{h,j}$
    if $g_{h,j} > \nu_{max}$ **then**
      $\Omega_{h+1,j+1}, \ldots, \Omega_{h+1,j+K} \leftarrow \text{section}(\Omega_{h,j}, K)$
      for $i \leftarrow 1$ **to** K **do**
        $\mu, \sigma \leftarrow \text{GP}(\mathcal{D}_t, K_D)$ // update_gp function
        $N \leftarrow N + 1$
        $x_{h+1,j+i} \leftarrow \text{center}(\Omega_n)$
        if $\mathcal{UCB}(x_{h+1,j+i}, \mu, \sigma) \geq f^+$ **then**
          $g_{h+1,j+i} \leftarrow f(x_{h+1,j+i})$
          $t \leftarrow t + 1$
        end
        else
          $g_{h+1,j+i} \leftarrow \mathcal{LCB}(x_{h+1,j+i}, \mu, \sigma)$
        end
        if $g_{h+1,j+i} > f^+$ **then**
          $f^+ \leftarrow g_{h+1,j+i}$
        end
        $n \leftarrow n + 1$
        $\mathcal{T}_n \leftarrow \{(x_{h+1,j+i}, f_{h+1,j+i}, \Omega_{h+1,j+i})\}$
      end
      $\nu_{\max} \leftarrow g_{h,j}$
    end
  end
end
return best of $x_{h,j}, g(x_{h,j})$

A.1.0.1 Annexe 1 : Sujet de stage

A.2 Captures d'écran

A.3 Formules chimiques

A.4 Extraits de code source

Contents

Tables and Figures

Algorithms

Glossary and Acronyms

Introduction **1**

1 Company's presentation **2**
1.1 INRIA history . . . 2
1.2 INRIA center of Lille University . . . 3
1.3 BONUS team . . . 3

2 Subject Definition **4**
2.1 Large Langage Models (LLM) . . . 4
2.1.1 Deep Neural Networks (DNN) . . . 5
2.1.2 Self Attention Mechanism . . . 6
2.1.3 LLM Architecture . . . 7
2.1.4 Fine-Tuning . . . 8
Parameter Efficient Fine-Tuning (PEFT) . . . 9
2.1.5 Review and taxonomy of LLMs . . . 10
LLMs Taxonomy . . . 10
LLMs review . . . 10
2.2 Auto-DNN . . . 11
2.2.1 Problem Formulation . . . 11
2.2.2 Neural Architecture Search (NAS) . . . 12
2.2.3 Hyper-parameter optimization . . . 13
2.2.4 Optimization Algorithms taxonomy . . . 13
Exploratory Methods . . . 14
Metaheuristic Approaches . . . 14
Partition Based Optimization (PBO) . . . 15
Surrogate-Model Based Optimization (SMBO) . . . 16
2.2.5 Parallel Optimization and High Performance Computing . . . 16
2.3 LLMs application to manufacturing context . . . 17
2.3.1 LLMs-based quality control . . . 17
2.3.2 LLMs-based supply chain management . . . 17
2.3.3 LLMs-based predictive maintenance . . . 18
2.4 Search problematic . . . 18

3 Methodology **19**
3.1 A Literature-Based Approach . . . 19
3.2 Blackbox Elaboration . . . 20
3.2.1 Fine-Tuning of the Model . . . 21
3.2.2 Evaluation of the model . . . 22
3.3 Search Space Definition . . . 22
3.4 Optimization Algorithms . . . 23
3.4.1 Bayesian Optimization (BO) . . . 24
3.4.2 Simultaneous Optimistic Optimization (SOO) . . . 25
3.4.3 Bayesian Multi Scale Optimistic Optimization (BaMSOO) . . . 26
3.5 Concrete Implementation . . . 27

3.6 experiments setup 28
3.7 Difficulties 28

4 Results **29**
4.1 Experiment Results 29
4.2 Article Publication 29
4.3 If This Work Were to Be Continued 30

5 Reflexion and Prospection **31**
5.1 Generalisation of the work 31
5.2 an industrial context 31
5.3 an intership, part of a bigger cursus 31

Conclusion et perspectives **32**

References

A Annexes **II**
A.1 Algorithm II
A.1.0.1 Annexe 1 : Sujet de stage II
A.2 Captures d'écran III
A.3 Formules chimiques III
A.4 Extraits de code source III