Tampere University

Nguyen Quang Duc

IMPROVING CODE QUALITY USING FINE-TUNING LARGE LANGUAGE MODELS

A Comparative Analysis of Fine-tuned and Benchmark Models for Code Generation Capability

Bachelor's Thesis
Faculty of Information Technology and Communication Sciences
Supervisor: Dr. Mika Saari
October 2024

ABSTRACT

Nguyen Quang Duc: Improving code quality using fine-tuning large language models
Bachelor's Thesis
Tampere University
Computing and Electrical Engineering
October 2024

Large language models has demonstrated significant capabilities in solving real-life problems by the means of generating human-like responses to input text in the similar format. However, these generic languages exhibit weakness when the context is not fully provided, resulting a need for model customization to enhance their performance in specialized fields. The fine-tuning is implemented using techniques inspired by machine learning methods. The two well-known fine-tuning methods are Supervised Fine-Tuning (STF), based on the concept of supervised learning, and Reinforcement Learning from Human Feedback (RLHF), firstly introducted by OpenAI [1]. The resource-intensive nature of RLHF makes it challenging to replicate the results achieved by Chat Generative Pre-trained Transformer (ChatGPT) models. Therefore, a new technique named direct preference optimization (DPO) has been introduced lately, incorporating similar concepts to RLHF while reducing required time and labor.

Given several studies have not addressed the improvement of DPO compared to other fine-tuning methods and its interplay with them, this thesis employs STF, DPO and their combination to fine-tune the base LLAMA 3.1 model, whose size is 8 billion parameters, in order to improve code quality generated by the LLMs. These fine-tuned models are then evaluated using Bi-Lingual Evaluation Understudy (BLEU) and Bidirectional Encoder Representations from Transformer (BERT) metrics, along with their code-specific derivatives. Despite the outcomes of the metrics cannot determine the most effective fine-tuning technique, they clearly highlight the significant influence of dataset utilized during the fine-tuning process. The repository containing the code and data for the research can be accessed via GitHub platform[1].

Keywords: Large Language Model, LLM, Fine-tune, Supervised fine-tuning, STF, Direct Optimization Preference, DPO, deep learning, neural networks

[1]The code and data are accessible at https://github.com/ducnguyenquang1112001/STF-DPO-Fine-Tuning

PREFACE

This research aims to fill the knowledge gap in the topic of fine-tuning the large language models for specific research purposes, particularly among the available bachelor's theses in Trepo, the research paper platform of Tampere University. This paper also serves as the bachelor's thesis in Computing and Electrical Engineering program at Tampere University.

My supervisor, Dr. Mika Saari, I would like to express my sincere gratitude for his constructive suggestion and remarks during the implementation of the research.

Lastly, I would like to thank Mrs. Zheying Zhang for her guidance and assistance with my questions throughout the writing of this paper.

Tampere, 27th October 2024

Nguyen Quang Duc

CONTENTS

LIST OF SYMBOLS AND ABBREVIATIONS

AI	Artificial Intelligence
AST	abstract syntax tree
BERT	Bidirectional Encoder Representations from Transformer
BLEU	Bi-Lingual Evaluation Understudy
BP	brevity penalty
ChatGPT	Chat Generative Pre-trained Transformer
CNN	Convolutional Neural Network
CodeBERT	Code Bidirectional Encoder Representations from Transformer
CodeBLEU	Code Bi-Lingual Evaluation Understudy
DF	Disfluency
DNN	deep neural network
DPO	Direct Preference Optimization
IDF	inverse document frequency
LLAMA	Large Language Model Meta AI
LLM	Large Language Model
LoRA	Low-Rank Adaptation
ML	Machine Learning
NN	neural network
NPL	Natural Language Processing
PEFT	parameter-efficient fine-tuning
PTQ	post training quantization
QAT	quantization aware training
QLoRA	Quantized Low-Rank Adaptation
RL	Reinforcement Learning
RLHF	Reinforcement Learning from Human Feedback
SPR	Successful Prediction Rate
STF	supervised fine-tuning

1. INTRODUCTION

The discussion of Artificial Intelligence (AI) has became increasingly popular in recent years. The use of AI has been emerging in a wide variety of fields. For instance, the use of AI in healthcare sector has been a part of disease diagnosis, treatment and management process in healthcare sector [2]. Another prominent application of AI lied in the prediction of future events such as weather forecast with enhanced accuracy compared to conventional methods in [3] or the projection of extreme climate events including heavy precipitation events in [4]. Since AI is a relatively broad field of study, there are several types of AI available specializing in different functionalities including reactive, limited memory, theory of mind and self-aware [5]. One of the most prominent uses of reactive AI is Large language model (LLM) which show signs of understanding text data input by humans and respond with useful output in the same format as input. After the release of ChatGPT by OpenAI in 2022 [6], which stands for Chat Generative Pre-trained Transformer, the LLM has been characterized by its capabilities to formulate a thorough and human-like conversation and assisting in reasoning and programming-related tasks.

Besides LLMs' ability to conduct a human-like conversation, the assessment of its output in the form of text using automatic essay scoring (AES) has shown high degree of validity and reliability in [7]. In addition, the code generation capability of LLMs is regarded as a significantly supportive element in software development tasks [8]. As a result, the concern of quality of code created by LLMs has captivated the spotlight. In particular, [9] examined the efficiently of programming solution provided by LLMs using HumanEval and MBPP metrics. The competence of LLMs in debugging is also challenged with customized benchmark called DebugBench in [10].

Research studies on these topics have inspired us to further explore the code generation potential of open-source available LLM as well as fine-tuned ones. This thesis illustrates the comparison of a benchmark LLM with other fine-tuned models using various techniques in terms of various metrics evaluating code quality based on provided code reference. The thesis aims to seek for the answers for the following research questions.

1. How can large language models (LLMs) be fine-tuned to improve their code generation capabilities in terms of quality?
2. How are fine-tuned LLMs using various fine-tuning methods compared to the origi-

nal baseline LLM in terms of code effectiveness?

3. Which fine-tuning technique out of examined ones is the most effective approach in enhancing LLMs' code quality?

The structure of the thesis is formulated as follows. In the next chapter, the theoretical background of the closely related topics is discussed, followed by the literature review on similar studies done by other researchers. Chapter 3 is planned to demonstrate the fine-tuning methods implemented in the paper with detailed configuration properties. Moreover, the chapter includes the description of the datasets in use along with their construction process. Chapter 4 reveals the comprehensive background of evaluation metrics and results of the experiment using the mentioned metrics. Furthermore, Chapter 5 holds discussion and analysis of results from the previous chapter to gain possible insights among the obtained data. Finally, Chapter 6 concludes the thesis with distilled comprehension from findings along with potential direction of future researches based on this thesis.

2. THEORETICAL BACKGROUND

This chapter introduces general theoretical background for the basic concepts, which are highly correlated with the research topic. These notions includes machine learning (ML), neural networks (NN), large language model (LLM) and fine-tuning in the models. After establishing foundational concepts, followed literature review give audience the general perspective on work conducted in the fields of AI.

2.1 Machine Learning

Machine learning (ML) is defined as a branch of computer science where machine shows its capability to imitate human's behaviour. This process could be done by making use of algorithm and available data to enable the machine to behave in a way that reduces the amount of error it made after each learning cycle. Therefore, when the machine has gone through the learning period, it tends to show signs of human-like intelligence, thus called artificial intelligence (AI), due to the its similar approach to how humans solve real-life problems. In simple terms, ML can be seen as "the field of study that gives computers the ability to learn without explicitly being programmed" by Arthur Samuel in [11].

The machine needs to get access to the data to be able to perform the learning process. Depending on the specific goal of the machine processing the data, ML can be categorized into multiple different types. One of the most important sort of ML is unsupervised learning, where algorithm is applied to search for underlying patterns among unlabeled dataset and group the provided dataset into various clusters [12]. This approach of learning could be considered useful if the problem is not fully understood and comprehensively defined at its initial state. In such case, unsupervised learning may play a crucial role in discovering the hidden causes or factors of the issues rather than making direct decisions from the input [13]. Then, the causes could be fed into another system, which owns a separate algorithm of ML, to formulate more concise and reliable output [13].

On the contrary, supervised learning, a different category of ML, is generally relevant in the context of training the machine to perform better in solving a well-known problem. Since the issue has been well studied, one could derive appropriate labels and their content of a certain dataset, and theses labeled dataset are then utilized to teach the machine to behave in a way that minimize deviation from patterns provided in the training

dataset. Typically, the dataset operated in supervised learning contains inputs and their correlated desired outputs, which is manually formulated in most cases [14]. Then, the machine has to automatically adjust its properties to generate the outputs from the inputs in the dataset so that the loss calculated with a function reduces over the training process [14].

Supervised and unsupervised learning are based on the assumption that each record in the dataset are independent to each other. A third type of ML, reinforcement learning (RL), takes an opposite approach where every records inside the dataset are interdependent, meaning past results can affect current and future prediction [15]. Reinforcement learning is commonly used to produce an agent, which makes decisions in a simulated environment toward a goal without any intervention of human [15]. To be able to achieve the predetermined goal, which is also a part of the environment, the agent must interact with the environment. Moreover, as human guidance is absent in the setting, the machine has to perform a wide variety of action and observe the following results. During the process, correct actions are rewarded while incorrect ones are punished. This trial-and-error strategy enables the machine to gravitate towards correct and optimal decisions to obtain the goal inside the environment. It is important to highlight that for the agent to learn from the past events in each interactive iteration, results of its actions in different time frames must be able to influence each other in a particular manner.

2.2 Neural Networks

In the last section of machine learning, the referred subjects of different ML types includes machine, system and agent. This part of the chapter elaborates on the concept of neural network (NN), which is the target subject of the mentioned machine learning methods. The definition of NN was heavily inspired by the biological structure of living organism's brains. Brain and nervous system of human mainly consists of neuron cells, which are responsible for sending nerve signal from one end of the cell to another [16]. Similar description could be found in the context of ML, where neural networks is a collection of artificial neurons connected to each other [17]. In fact, the first version of artificial neuron was an imitation of biological neuron in a form of mathematical model proposed by McCulloch and Pitts in 1943 [18]. In 1957, an improved variant of artificial neuron called perceptron was introduced by Frank Rosenblatt [19]. The structure of a single perceptron encompasses various elements: weighted inputs, summation function, activation function, output and bias [19].

As can be seen from Figure 2.1, each input x_i is multiplied with the correlated weight w_i before all the weighted inputs are passed forward to the summation function to calculate the sum of these inputs. Additionally, bias is added to the outcome to shift the output of the perceptron. Lastly, the activation function receives the result and decides whether the

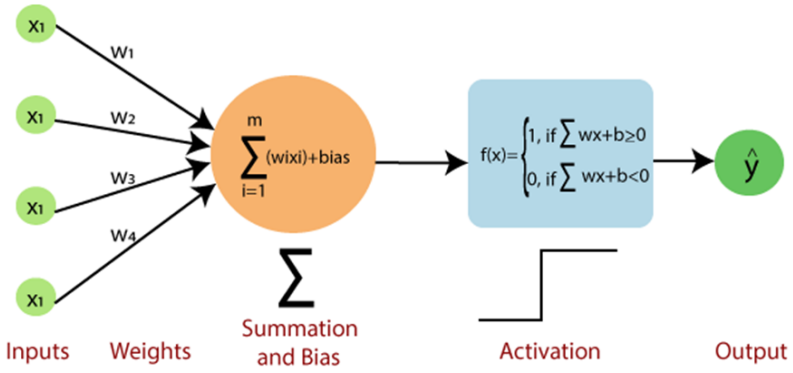

Figure 2.1. *Structure of a perceptron [20].*

neuron is activated [21]. In the context of the figure, the perceptron would be triggered if the outcome is greater than or equal to zero, and the neuron remains inactivated when the result is smaller than zero.

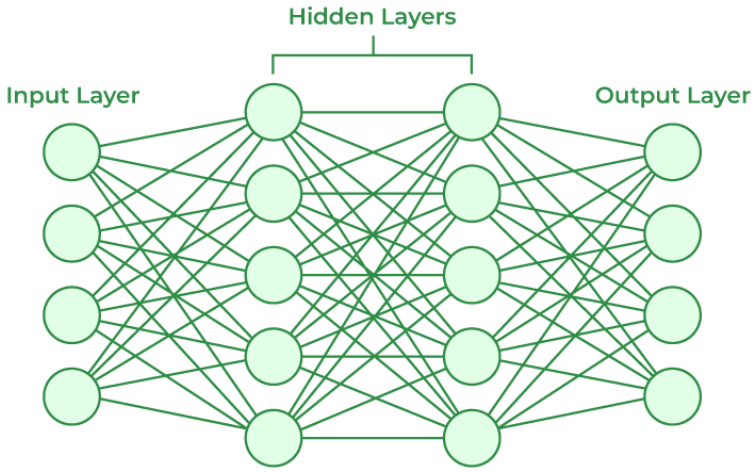

Figure 2.2. *A simplified structure of a neural network [22].*

Neural network is interpreted as a network, as its name suggests, of artificial neurons, which are perceptrons, organized in a particular manner. Specifically, NN consists of multiple layers of perceptrons, where each perceptron in one layer connects to every perceptron of the next layer, forming a fully connected structure. These layers are divided into three different types: input, hidden and output layers as shown in Figure 2.2. The input layer is the first layer in the network and acts as the carrier of the input data before it is passed into the next layer. Therefore, there is no connection to the input layer from the previous layer, and the number of dimensions in the input layer is defined by the size of input data. The next group of layers are hidden ones. The number of layers inside the hidden layers is referred as the depth of the network itself [23]; thus, if a network owns more than one hidden layer, it could be considered as a deep neural network (DNN).

These hidden layers process the input data from the input layer with mathematical functions of their perceptron. Eventually, the calculated outcomes of the hidden layers are transferred to the output layer, where each perceptron represents a possible output of the network. This suggests that the size of output layer is highly subjected to predetermined requirement of the problem intended to be solved by the use of NN.

2.3 Large Language Model

One of the most prominent types of DNN in the field of natural language processing (NLP) is a large language model (LLM) which shows signs of understanding text data input by humans and responds with useful output in the same format as input [24]. To achieve such competence, LLM has been trained using a relatively huge amount of data in various complex contexts. The diversity of text data helps the LLM to have basic comprehension of grammar and the use of words, phrases and sentences in multiple circumstances. Nonetheless, the exceptional language proficiency of LLM is attributed to the adoption of a state-of-the-art architecture named Transformer.

The design of Transformer consists of two main components, which are encoder and decoder. Encoder translates input text into embedded description that contains useful information from the original content [26]. Meanwhile, decoder receives the outcome of the encoder and transforms it into another format [27]. In Figure 2.3, it appears that before being passed to the encoder and decoder, inputs and outputs must be embedded into numerical values by embedding layers and combined with positional encoding. Positional encoding helps the model recognise different positions of tokens inside the sequence by constructing sine and cosine functions having distinctive frequencies [25]. Then, three separate vectors, including queries (Q), keys (K) and values (V), are calculated with the use of weighted sum and continue to go through Scaled Dot-Product Attention belonged to Multi-Head Attention in the encoder and masked one in the decoder [25]. The Scaled Dot-Product Attention function can computed as follows: [25]

$$Attention(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.1}$$

The attention function enables the model to perform self-attention, meaning the model focuses on different parts of the input to predict the output [28]. Therefore, in the encoder, the model can find self-attention scores of input tokens and understand the relationships between tokens in the sequence; however, the first attention layer of the decoder is completely masked, forcing the model to make prediction based only on the tokens on the left of current position. This mechanism makes sure the model is not able to get access to future tokens on the right of the position during decoding process. Residual connections are added to the output of attention mechanism, allowing the data to maintain intact

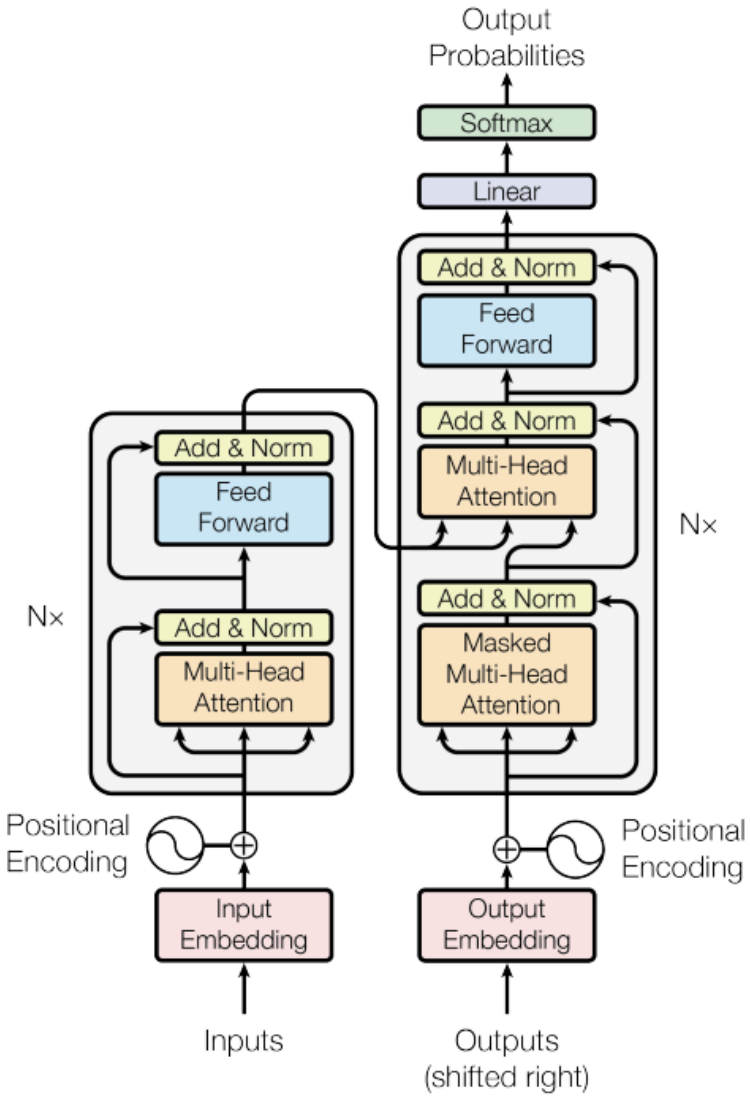

Figure 2.3. *Architecture of a Transformer [25].*

and move to the latter part without being modified by certain layers in the network [29], and normalization could take place in the connection outcome to improve the processing speed and performance. Then in the encoder, the result is fed to a feed forward neural network with residual connection and normalization applied. The next stage is called cross attention where the solution of encoders is concatenated into a vector and passed to multiple decoders. In particular, queries (Q) from decoders interact with keys (K) from encoders, using the dot product and a soft-max operation, to find the attention weights, which later applied to values (V) of the encoders. Combining sequences from two different types of components allows the decoders to pay attention to the data of inputs while inferring the outputs [30]. The residual connection and normalization are also implemented to the multi-head attention layer of the cross-attention stage. Similar to the encoders, another feed forward network, with residual connection and normalization attached to it, receives the product of the previous layer and yields the final values, which is further pro-

cessed through linear and soft-max layers to get the probability distribution of considered vocabularies.

2.4 Fine-tuning in Large Language Models

Although the open-sourced LLMs has shown to perform exceptionally in a wide variety of fields in terms of general knowledge and problem-solving, there are needs to customize the LLMs in specialized and downstream fields in an attempt to yield better results in pre-determined assessment metrics, which are considered crucial in such fields. In fact, OpenAI's third generation GPT is modified and fine-tuned into two different derivatives including "text-davinci-002" and "code-davinci-002", which exhibits improved performance in text and code generation respectively as shown in [31]. Similarly, the researches on the use of customized LLMs has started in multiple fields. In particular, Fine-tuning has been applied into a LLM to generate commentary during a football match in real time [32]. In [33], a fine-tuned GPT model has been utilized to conduct structured reviews on chemical-related risks, such as bisphenol A. For future event prediction in specific fields, the practice of fine-tuning a LLM during the prediction process has been indicated to reach satisfactory performance in wind power generation [34] or even excel accurate diagnosis rate for fault occurrence in [35]. In the field of psychology, fine-tuned LLMs identify patterns in patiences' speeches where linguistic disturbances exist with two custom evaluation metrics, Successful Prediction Rate (SPR) and Disfluency (DF) [36].

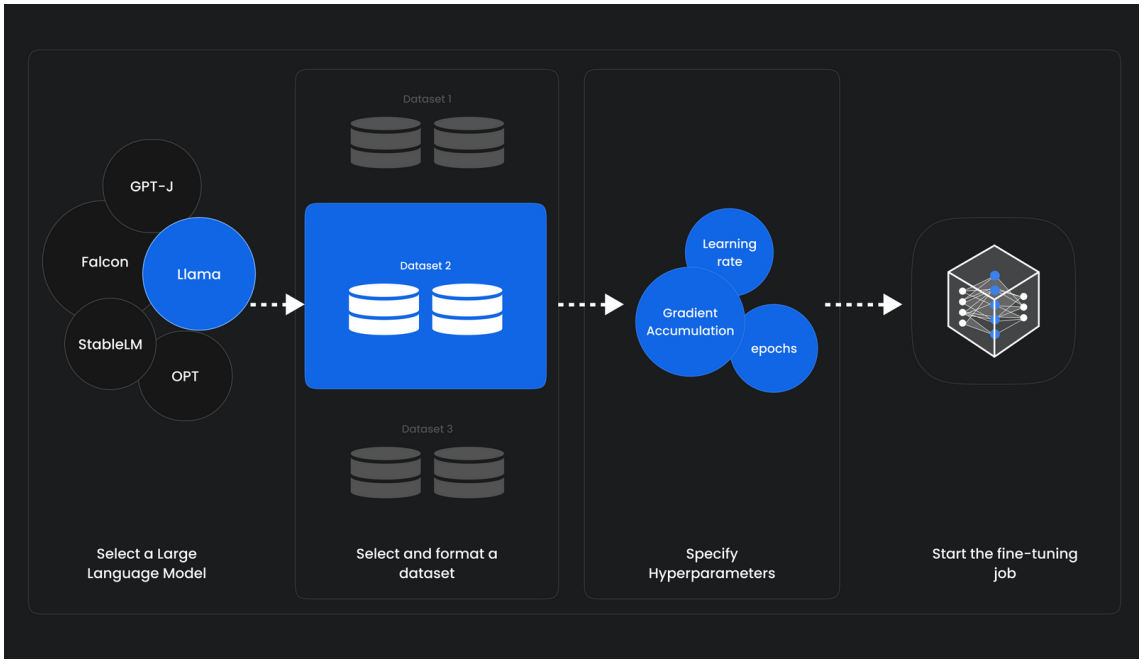

Figure 2.4. *Steps of fine-tuning process [37].*

Fine-tuning a LLM can be described as an additional process of training of an existing LLM, which has been trained using extensive data and publish by a third-party organiza-

tion such as OpenAI and Meta [38]. Therefore, to implement the fine-tuning process, an pre-trained LLM with desirable capabilities and performance must first be selected. Possible candidates for this step are LLMs provided by creditable establishments, including Chat Generative Pre-trained Transformer (ChatGPT), Bidirectional Encoder Representations from Transformers (BERT) and Large Language Model Meta AI (LLAMA). Similar to original training process, the availability of dataset is also crucial. Since the chosen model is generic in most cases, the content of dataset has to be highly customized and related to the targeted tasks. To further improve the quality of custom dataset, a number of data pre-processing tasks, encompassing data cleaning, transformation, reduction and integration, could be carried out [39]. Before the fine-tune begins, there are a multitude of fine-tuning techniques to be selected, and it is noteworthy that the structure and values of custom dataset is greatly dependent on the chosen type of fine-tuning. Moreover, hyperparameters such as the number of epochs, learning rate and gradient accumulation steps of training function can be personalized according to the preferences of the intended tasks. During the fine-tuning, weight values of internal parameters within the model are adjusted according to the dataset so that it could perform better in the specialized jobs.

2.5 Related works

In this section, challenges in fine-tuning LLMs recognized by multiple studies will be presented before the examination of common fine-tuning methodologies with their distinctive characteristics, and lastly, different research gaps are identified and elaborated in details after the illustration of mentioned themes.

Most studies have stated that the performance of fine-tuned LLMs ranges from desirable to significantly improved results in comparison with the benchmark models. Especially, the evidence in [33] clearly shows that the fine-tuned version of a GPT-3 model surpasses the available "text-davinci-002" model and even the more recent and advanced "text-davinci-003" model. In spite of LLMs' robust performance in domain-specific tasks, a number of challenges have been identified in other researches. For instance, the generation and classification of fine-tuned LLMs behave differently in novel tasks and domains; however, the situation could be improved by letting the model learn with the context-related method in fine-tuning process [40]. Additionally, [41] mentions that the use of small dataset during the fine-tuning of the examined model could possibly show signs of instability. To elaborate more about the instability, LLMs, which are trained with a large amount of dataset, could suffer from the overfitting when fine-tuning using a limited dataset of the target tasks [42]. Regrading the drawback of fine-tuning with low-resource dataset, the targeted tasks of these customized models in specialized fields tends to possess a fairly limited amount of ready-to-use dataset due to its nature, not to mention that the access to data might be limited in some sensitive fields. To compensate for the lack

of available dataset, more data could be made with annotation though it may increase the cost and time required to fine-tune the models. In terms of hyperparameters, the traditional setting of learning rate in training standard deep neural network (DNN) may not be optimal for the fine-tuning process of LLM, which is a subset of DNN specializing in text generalization [43]. This may require different policies of learning rate in fine-tuning the LLMs so that it could avoid the suboptimal performance.

Despite fine-tuned LLMs has been a part of a wide variety of applications in various fields, multiple distinctive methods are implemented to customize these models. These methodologies can be categorized based on their advantages over others or their notable characteristics. One prominent method is instruction-based tuning characterized by human intervention to make LLMs' responses matched with human's anticipation. In this category, supervised fine-tuning (SFT) appears to be the most widely used technique which conducts the model customization with a large dataset labeled according to a specific task [44]. Since the instruction tuning incorporates pairs of instruction and output that are corresponded to each other as the main training dataset, this practice is promised to keep the model's behavior in control and expectation of human [44]. Another approach that utilized guidance from human to perform fine-tuning is reinforcement learning from human feedback (RLHF) firstly introduced by OpenAI, the same organization that released GPT models, in 2017 [45]. Although the term was recently invented, it remained a part of reinforcement learning (RL) in machine learning (ML), a boarder concept describing an idea where a model could be trained using feedback from environment interaction seen as reward or punishment [45]. Different from SFT fine-tuning the models with instruction-based pre-training dataset, RLHF modifies the LLMs based on feedback directly from human instead of traditional previously defined metrics and gradients calculated from loss functions [45]. The feedback given by individuals, who are experts in task-related fields, clearly indicates the incentive and preference towards a specific set of results, which, in turn, manipulates the LLMs to act in favor of human's presumption.

Another kind of model optimization technique is called parameter-efficient fine-tuning (PEFT), which only requires modifying a smaller and thus more efficient number of parameter of the fine-tuned LLMs compared to other approaches. Adapter tuning, a type of parameter-efficient method, insert a randomly initialized adaptive layer, serving as a "bottleneck", inside the targeted network instead of tuning the top layer of the neural network [46]. Moreover, the weight of original network remains unchanged rather than being retrained together with the newly introduced top layer [46]. Due to the key trait of adapter layer, which only contains a small amount of parameters [46], this suggests that there are slight increase in neural network size after each fine-tuning process and significant reduction in parameter number required to be updated. In addition to the slow rise in size, the unaltered part of network, consisting of neurons with original weights, could be a common trait between different models adjusted for several tasks [46].

The memory requirement of available LLMs is considered demanding, for example LLaMA with parameter size ranging from 7 billion to 65 billion [47], even when memory demand has been significantly eased by the use of PEFT in fine-tuning process. This leads to the introduction of quantization technique, whose the primary goal is to reduce the inherently large size of a network. To complicate further, there are two main approaches to quantize a model: post training quantization (PTQ) and quantization aware training (QAT). PTQ is seen as a more straightforward way than the other technique as it does not require re-training the model, and experts have the option to quantize only weights or both weights and activation functions from CPUs' float point to 8-bit arithmetic [48]. To achieve higher precision as shown in [48], the adoption of QAT is recommended. Specifically, quantization operations are simulated on weights and activation functions while the original float values of weights are kept un-quantized to be updated with gradients, ensuring a higher accuracy in result values [49]. In [48], the quantization method had been implemented on a convolutional neural network (CNN) to yield noticeable results including the improvement of model's processing speed varying from two times to three times along with the possibility of model size decrease up to a factor of four.

Though the number of researches on fine-tuned LLMs increased dramatically in the last few years, it is inevitable that there are still gaps in our understanding of fine-tuning LLMs. One of them is the limited amount of studies on the relationship between LLMs' properties (models' size, size of pre-training and fine-tuning dataset, parameter amount, etc.) and the performance of fine-tuning. [50] has discovered the connection between the fine-tuning data size and other scaling traits of LLMs, but the studies assume its lack of theoretical groundings and experimental scenarios. Also in [50], the joint association between scaling factors of examined tasks is non-linear; hence, the authors suggest a specific fine-tuning method cannot be intuitively assigned to a certain type of tasks.

3. METHODOLOGIES

This chapter analyzes different methods of fine-tuning applied to the selected model during the research. Theoretical background and properties are carefully depicted for each method before the details of base model and dataset description are revealed, where multiple traits of the model and datasets used in each fine-tuning technique is mentioned.

3.1 Parameter Efficient Fine-Tuning (PEFT)

To display exceptional performance in natural language processing (NLP) tasks, open-sourced generic large language models (LLMs) are constructed with from billions up to trillions of parameters. For example, the parameter size of LLAMA models released by Meta AI ranges from 7 billion to 65 billions [47]. As a matter of fact, the LLM targeted for fine-tuning in this study is the Meta's LLAMA 3.1, which has 8 billion parameters and supports multiple languages, including English, German, French, Italian, Portuguese, Hindi, Spanish, and Thai [51]. Normally, every parameter of a full-precision LLM is represented using standard 32-bit floating-point numbers, which requires a considerable amount of memory space and computational power to fine-tune; thus, quantization method, which decreases the bit number utilized to symbolize a parameter within a model, is implemented to the original LLAMA 3.1 [52]. In detail, the 4-bit prevision quantized version of LLAMA 3.1 model serves as the primary LLM used throughout the fine-tuning processes in this research, which demands less memory footprints and inference time [52]. Nonetheless, the enormous number of parameters in the model poses as a challenge since modifying the weight of parameter-rich model is considered computationally intensive and time-consuming. To address this issue, Parameter Efficient Fine-Tuning (PEFT) was introduced, which trains merely a subset of parameters within the models while achieving performance similar to the full fine-tuning process [53]. Moreover, custom dataset has smaller size than original one utilized in the generic LLM training process, increasing the probability of overfitting during the full fine-tuning process [53].

PEFT consists of several categories of fine-tuning methods distinguished by their approaches, including additive fine-tuning, partial fine-tuning, reparameterized fine-tuning, hybrid fine-tuning and unified fine-tuning [53]. Out of these fine-tuning techniques, Low-Rank Adaptation (LoRA), which belongs to the category of reparameterized fine-tuning,

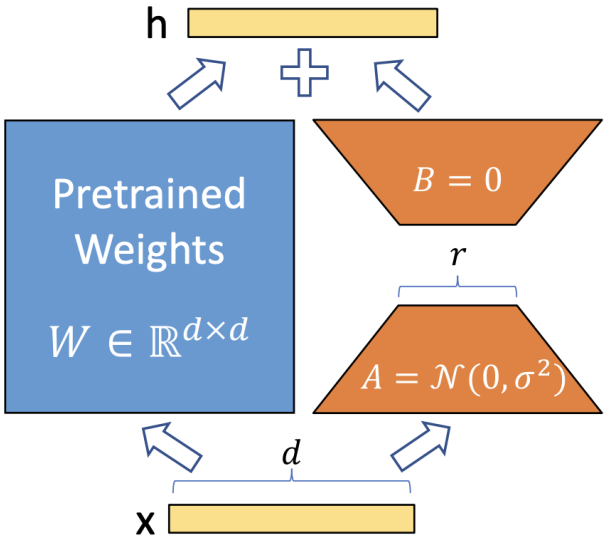

Figure 3.1. *Process of Low-Rank Adaptation (LoRA) fine-tuning [54].*

has been selected as the primary method for this research. This category of fine-tuning is characterized by the use of low-rank transformation to enable the reduction of trained parameters [53]. Especially, LoRA has been shown to decrease the amount of trained parameters by 10,000 times and required memory of graphics processing unit (GPU) by 3 times compared to the counterpart Adam fine-tuning method in GPT-3 model whose parameter size is 175 billions [54]. During LoRA fine-tuning, the model's pre-trained weights W are kept intact while the fine-tuned weights ΔW, which are added to the original weight after computed, are low-rank decomposed into 2 different matrices A and B as seen in Figure 3.1 [54]. After the decomposition, the total number of trainable parameters, which is the sum of elements in matrices A and B, is remarkably reduced in comparison with the full weight difference matrix ΔW. To balance between the fine-tuning efficiency and the preservation of information inside the weight change matrix ΔW, the rank r of the low-rank decomposition is defined as 16 in all fine-tuning process in the paper. Applying LoRA method to fine-tune a quantized model is known as QLoRA (Quantized Low-Rank Adaptation), which also opens up new possibilities for fine-tuning parameter-rich LLMs with limited resources [55].

3.2 Supervised Fine-tuning (STF)

One of the examined fine-tuning techniques in the paper is supervised fine-tuning (STF). Inspired by the concept of supervised learning introduced in Chapter 2, labeled data in custom dataset is utilized to fine-tune a LLM, adjusting its parameters so that the model gain knowledge in a specific domain, or weight of a particular context could be emphasized within the model, increasing its chance to solve niche tasks. Particularly, the dataset

used in STF typically consists of prompt-answer pairs that the model is trained to emulate during the fine-tuning process [56]. To avoid impact from other programming languages, we aims at Python for fine-tuning and evaluating the model's code quality, and to compile the dataset, we retrieved *'iamtarun/python_code_instructions_18k_alpaca'* dataset from Hugging Face[1], a sharing platform of ML models and datasets. This dataset contains a roughly 18,600 Python-related data rows, each encompassing three different fields: instruction, input and output. These fields are highly compatible with prompt based on Alpaca format, where an contextual instruction called pre-prompt is called before the mention of the fields. The instruction and output fields provides the clear descriptions of tasks and the following responses that the model is expected to produce respectively. Meanwhile, the input fields are optional and can be used to supplement the model with additional or contextual information to enhance its performance. Since the original dataset possesses only one training split, it was manually divided into a training set of 16,750 rows and a validation set of 1,862 rows. To ensure the testing set does not overlap with the training and validation sets, we randomly selected 100 rows from a separate dataset named *'flytech/python-codes-25k'* from the Hugging Face platform[2] and created a testing set used across all three concerned fine-tuning cases.

3.3 Direct Preference Optimization (DPO)

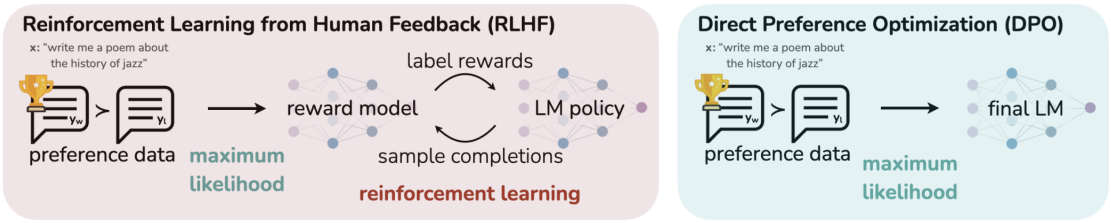

Figure 3.2. *Comparison between Reinforcement Learning from Human Feedback (RLHF) and Direct Preference Optimization (DPO) [57].*

Reinforcement Learning from Human Feedback (RLHF) has shown improvement in model's performance compared to supervised learning by utilizing a reward model trained with human comparison, which optimizes the policy during the reinforcement learning process of the pre-trained model [1]. However, it is clear that the implementation of RLHF is significantly more time-consuming and computationally intensive due to the additional training of reward model, typically initialized with a copy of the original target model. Furthermore, to mitigate the risk of negative exploitation of the reward structure, the reward policy has to be defined to prevent the model from deviating considerably with the following function: [1]

[1]The dataset is accessible at https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca
[2]The dataset is accessible at https://huggingface.co/datasets/flytech/python-codes-25k

$$R(x, y) = r_\theta(x, y) - \beta \log \left[\frac{\pi_\phi^{\mathsf{RL}}(y|x)}{\pi^{\mathsf{SFT}}(y|x)} \right] \tag{3.1}$$

Another fine-tuning method under investigation is Direct Preference Optimization (DPO), which selects preferred responses of models directly from its broad knowledge base without relying on a reward model and reinforcement learning, thereby reducing the complexity of the fine-tune process [57]. In RLHF, the optimal policy is derived from multiple iterations of training loop while the DPO method calculates the optimal policy in closed form, meaning that it could be demonstrated explicitly as a function as follows [57].

$$\pi_\theta(y \mid x) = \frac{\pi_{\mathsf{ref}}(y \mid x) \cdot \exp\left(\frac{r(x,y)}{\beta}\right)}{\sum_{y'} \pi_{\mathsf{ref}}(y' \mid x) \cdot \exp\left(\frac{r(x,y')}{\beta}\right)} \tag{3.2}$$

With the established policies, we are able to transition from calculating the loss function centered around reward functions to deriving the following loss function that is grounded in the defined policies [57].

$$\mathcal{L}_{\mathsf{DPO}}(\pi_\theta; \pi_{\mathsf{ref}}) = -\mathbb{E}_{(x,y_w,y_l)\sim D} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w|x)}{\pi_{\mathsf{ref}}(y_w|x)} - \beta \log \frac{\pi_\theta(y_l|x)}{\pi_{\mathsf{ref}}(y_l|x)} \right) \right] \tag{3.3}$$

The loss function plays a crucial role in aligning the the model's policy π_θ with reference policy π_{ref} reflecting the probability of human-preferred data [57]. In simple terms, it increases the probabilities of positive example y_w and reduces those of negative instance y_l while ensuring the model remains close to the initial state.

Because there is a lack of readily available datasets eligible to DPO fine-tuning method specifically for Python programming language, it is necessary to manually build a new dataset using the available resources, which was uploaded to the Hugging Face platform[3] and named *'quangduc1112001/python-code-DPO-fine-tune'*. Similar to standard datasets utilized in RLHF, this dataset comprises a total of 2,000 rows of data, with each row consisting of three distinct fields: prompt, chosen and rejected. Among these properties, the prompt and chosen fields are randomly picked from the same dataset used in STF[4], known as *'iamtarun/python_code_instructions_18k_alpaca'*, while the rejected field is obtained from the inference of the base LLAMA 3.1 model based on the corresponding prompt. Besides, there are two separate splits within the dataset, including training and validation sets, whose row numbers are 1,800 and 200 respectively.

[3]The dataset is accessible at https://huggingface.co/datasets/quangduc1112001/python-code-DPO-fine-tune

[4]The dataset is accessible at https://huggingface.co/datasets/iamtarun/python_code_instructions_18k_alpaca

Apart from the two examined fine-tuning techniques, we also show an interest in exploring the effects caused by the combination of the two methods. Therefore, the final fine-tuning scenario involves merging STF and DPO methods, meaning that the base model which underwent the STF process continued to be fine-tuned with DPO techniques, using the same dataset as in the DPO-only case.

4. RESULTS AND EVALUATION

This chapter delves into the definition and concepts behind the metrics evaluating fine-tuned models in various scenarios. Then, the results of these evaluations is carefully presented and illustrated in detail.

4.1 BLEU and CodeBLEU Metrics

Quality control and evaluation are integral aspects of training and deploying well-performing LLMs. These metrics enable us to be aware of the extent of improvement or decline of the model's performance after training or fine-tuning; thus, the relative effects of each fine-tuning techniques are analyzed to identify the most optimal solution. As the nature of machine learning algorithm are explained in Chapter 2, the responses of a language model are not static and subject to changes. Although estimating the quality of LLMs' outputs is clearly time-consuming and labor-intensive, specialized metrics, commonly referred to as LLM evaluation metrics, have to be in use to assess the fine-tuned LLMs. The categories of these metrics vary based on the assigned architectures. Two types of assessment metrics inspected in the experiment include statistical scorers and model-based scorers. Statistical scorers utilize merely statistical methods to calculate the difference between the actual and expected LLM's responses, making its results highly reproducible and reliable [58]. In contrast, the outcomes of model-based models scorers are computed by a standalone LLM, allowing the evaluation process to save required time and resources [58].

One of the most widely-used statistical scoring methods for machine translation tasks is the Bi-Lingual Evaluation Understudy (BLEU) metric family. The BLEU score computation depends on the number of matching comparisons between n-grams of the model's outputs and n-grams of the expected responses [59], where an n-gram is understood as a continuous sequence of n words in a text. To prevent the n-gram precision score from being overrated when incorrect duplicated words in model's s appeared more frequent than ones in the reference outputs, modified n-gram precision, which ensures the count of matching n-grams does not exceed their occurrences in reference texts, is demonstrated with the following function [59]:

$$p_n = \frac{\sum_{C \in \text{Candidates}} \sum_{\text{n-gram} \in C} \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{C' \in \text{Candidates}} \sum_{\text{n-gram}' \in C'} \text{Count}_{\text{clip}}(\text{n-gram}')} \tag{4.1}$$

Using Equation 4.1, the modified precision for unigrams, bigrams, trigrams and 4-grams is calculated and combined to yield the cumulative 4-gram BLEU score, also known as BLEU-4, in the subsequent equation [59].

$$\text{BLEU-4} = \text{BP} \cdot \exp\left(\sum_{n=1}^{4} w_n \log p_n\right) \tag{4.2}$$

where brevity penalty (BP) assure the highly preferred outputs align with the references in terms of text length, word choice and order, as defined by the function below [59].

$$\text{BP} = \begin{cases} 1 & \text{if } c > r \\ e^{(1-\frac{r}{c})} & \text{if } c \leq r \end{cases} \tag{4.3}$$

However, the BLEU metric does not take into consideration unique characteristics of codes, such as limited number of words, language structure and different instructions, as it was primarily constructed to evaluate the NLP tasks [60]. Thus, another derivative of BLEU metric called Code Bi-Lingual Evaluation Understudy (CodeBLEU) is suggested to overcome these confinements. Specifically in Figure 4.1, there are a total of four elements, which are the BLEU score, the weighted n-gram match, the syntactic abstract syntax tree (AST) match and the semantic data-flow match, combined according to weighted contribution [60].

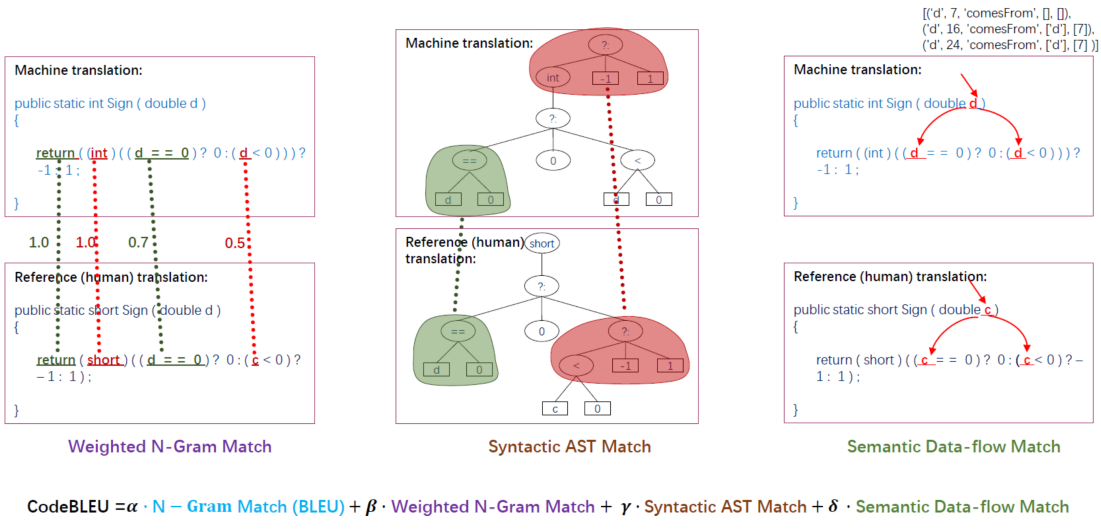

Figure 4.1. *An overview of the CodeBLEU evaluation metric structure [60].*

Unlike from natural language, programming language consists of a limited set of words,

with keywords such as 'private', 'string' and 'int' being crucial as they are integral parts of the language's syntax. Weighted n-gram match takes the characteristic into account by introducing the weights μ_n^i to n-gram precision calculation in Equation 4.1 as follows [60].

$$p_n = \frac{\sum_{C \in \text{Candidates}} \sum_{\text{n-gram} \in C} \mu_n^i \cdot \text{Count}_{\text{clip}}(\text{n-gram})}{\sum_{C' \in \text{Candidates}} \sum_{\text{n-gram}' \in C'} \mu_n^i \cdot \text{Count}_{\text{clip}}(\text{n-gram}')} \tag{4.4}$$

It is also noteworthy that considering the significance of keywords in programming languages, the weight μ_n^i of keywords is set 5 times higher than that of other words [60]. Beside computation at the level of text sequence, an abstract syntax tree (AST) is incorporated to demonstrate the syntactic structure of the code in the form of a hierarchical tree [60]. Afterward, the trees of generated and reference code are decomposed into all possible sub-trees, which are used to identify the syntactic AST match measuring quality of code through a syntactic lens, as shown below [60].

$$\text{Match}_{\text{ast}} = \frac{\text{Count}_{\text{clip}}(\text{T}_{\text{cand}})}{\text{Count}(\text{T}_{\text{ref}})} \tag{4.5}$$

Another prominent feature of programming languages is the use of variables, whose misuse is hardly detected using normal or weighted n-gram precisions and syntactic AST match. To rectify this issue, one could approach the code structure in a semantic level to gather data flow of the used variables within the program as seen in [61]. Building on the earlier trees generated using the AST method, GraphCodeBERT [61] is employed to extract the variables and semantic relations from the leaves of the tree, which are later injected into the tree in a form of data-flow graph $G(\text{C}) = (V; E)$ [60]. To achieve the uniformity and simplicity of comparison task, location and original names of variables are ignored and normalized respectively [60]. This enables us to easily determine the number of matching data-flow items in terms of normalized variable names and their relationship, which plays a pivotal role in computing the score of semantic data-flow match as defined in Equation 4.6 [60].

$$\text{Match}_{\text{df}} = \frac{\text{Count}_{\text{clip}}(\text{DF}_{\text{cand}})}{\text{Count}(\text{DF}_{\text{ref}})} \tag{4.6}$$

where DF_{cand} represents the data-flow items in the generated response that are matched with DF_{ref}, which represents the items in the reference code [60].

4.2 BERT and CodeBERT Metrics

As clearly demonstrated in Section 4.1, the BLEU metric family is solely dependent on statistical methods to assess relevance between reference and produced outputs; thus,

the lack of contextual understanding and semantic reasoning makes the category of statistical scorers, including BLEU and CodeBLEU metrics, less effective for complicated and lengthened responses from LLMs although there is also concern related to the bias of LLM-based evaluator toward its own generated output during self-evaluation process [62]. To tackle this issue, we introduce a new metric family in our research, including Bidirectional Encoder Representations from Transformer (BERT) and Code Bidirectional Encoder Representations from Transformer (CodeBERT) metrics, which belong to both categories of statistical and model-based scorers.

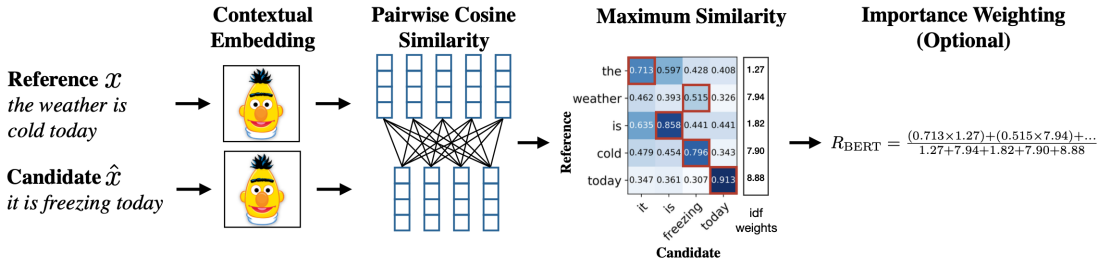

Figure 4.2. *Process of calculating recall score in BERT metric [63].*

As illustrated in Figure 4.2, BERT metric firstly converts the human reference and LLM-created text into contextually embedded vectors with the help of BERT model as its name suggests [63]. Then, the vectors are normalized and compared with cosine similarity using the equation as follows [63].

$$\frac{\mathbf{x}_i^\top \hat{\mathbf{x}}_j}{\|\mathbf{x}_i\|\|\hat{\mathbf{x}}_j\|} \tag{4.7}$$

where $\mathbf{x}_i$ denotes the embedding vector of token x_i in the reference text, and $\hat{\mathbf{x}_j}$ is the corresponding embedding vector for the token in generated sentence. The recall is calculated by matching each token of x with the highest-scoring token of $\hat{x}$ while the precision is computed with matches between every token in $\hat{x}$ and the token in x with the highest cosine similarity [63]. From the recall and precision, F_1 score can be found in the function provided below [63].

$$R_{\text{BERT}} = \frac{1}{|x|}\sum_{x_i \in x} \max_{\hat{x}_j \in \hat{x}} \mathbf{x}_i^\top \hat{\mathbf{x}}_j \tag{4.8}$$

$$P_{\text{BERT}} = \frac{1}{|\hat{x}|}\sum_{\hat{x}_j \in \hat{x}} \max_{x_i \in x} \mathbf{x}_i^\top \hat{\mathbf{x}}_j \tag{4.9}$$

$$F_{1\text{BERT}} = 2\frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{4.10}$$

To prevent unusual words from being considered more significant than typical ones in the similarity calculation, inverse document frequency (IDF) of token w can be integrated into recall and precision scores [63].

$$\text{IDF}(w) = -\log \frac{1}{M} \sum_{i=1}^{M} \mathbb{I}[w \in \hat{x}^{(i)}] \tag{4.11}$$

In Equation 4.11, M is the total amount of reference sentences, and $\mathbb{I}[\cdot]$ is the indicator function.

Regardless of advancement and competence of BERT, it still encounters similar s due to its inherent design and difference between natural and programming languages in code evaluation, as discussed in Section 4.1. CodeBERT score has addressed these shortcoming through several important modifications. In particular, instead of utilizing the standard BERT model to encode the tokens, the selected base model, CodeBERT denoted as β, is trained on a resourceful dataset for 1,000,000 steps for each programming language, ensuring the robustness of assessment performance [64]. To further enhace the contextual understanding of the code, the tokenized context x is concatenated to tokenized reference code y^* and and generated code $\hat{y}$ in Equation 4.12 before the contextual embedding process [64].

$$\begin{aligned} \tau_\beta(x \cdot y^*) &= \langle x_1, \ldots, x_k, y_1^*, \ldots, y_m^* \rangle \\ \tau_\beta(x \cdot \hat{y}) &= \langle x_1, \ldots, x_k, \hat{y}_1, \ldots, \hat{y}_n \rangle \end{aligned} \tag{4.12}$$

After being embedded into context information, the result vectors of reference and generated code, which are $\langle \mathbf{x}_1, \ldots, \mathbf{x}_k, \mathbf{y}_1^*, \ldots, \mathbf{y}_m^* \rangle$ and $\langle \mathbf{x}_1, \ldots, \mathbf{x}_k, \hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_n \rangle$ respectively, is masked with correlated masks $\mathbf{m}^*$ and $\hat{\mathbf{m}}$ to remove non-alphanumeric tokens for more precise similarity calculation in the following stage [64]. As a result, the functions will demonstrate the recall and precision scores of CodeBERT metric as follows.

$$\begin{aligned} R_{\text{CodeBERT}} &= \frac{1}{|\mathbf{y}^*[\mathbf{m}^*]|} \sum_{\mathbf{y}_i^* \in \mathbf{y}^*[\mathbf{m}^*]} \max_{\hat{\mathbf{y}}_j \in \hat{\mathbf{y}}[\hat{\mathbf{m}}]} sim(\mathbf{y}_i^*, \hat{\mathbf{y}}_j) \\ P_{\text{CodeBERT}} &= \frac{1}{|\hat{\mathbf{y}}[\hat{\mathbf{m}}]|} \sum_{\hat{\mathbf{y}}_j \in \hat{\mathbf{y}}[\hat{\mathbf{m}}]} \max_{\mathbf{y}_i^* \in \mathbf{y}^*[\mathbf{m}^*]} sim(\mathbf{y}_i^*, \hat{\mathbf{y}}_j) \end{aligned} \tag{4.13}$$

where $\mathbf{y}[\mathbf{m}]$ is defined as the result of applying the mask $\mathbf{m}$ to the vector $\mathbf{y}$, and Equation 4.14 provides the definition of the similarity function, denoted as $sim(\cdot)$ [64].

$$sim(\mathbf{y}_i^*, \hat{\mathbf{y}}_j) = \frac{\mathbf{y}_i^{*\top} \cdot \hat{\mathbf{y}}_j}{\|\mathbf{y}_i^*\| \cdot \|\hat{\mathbf{y}}_j\|} \tag{4.14}$$

In addition to F_1 score, CodeBERT also returns the F_3 score where recall score is emphasized more than precision counterpart as shown in the subsequent equation [64].

$$F_{3\text{CodeBERT}} = \frac{10 \cdot P_{\text{CodeBERT}} \cdot R_{\text{CodeBERT}}}{9 \cdot P_{\text{CodeBERT}} + R_{\text{CodeBERT}}} \tag{4.15}$$

4.3 Experimental Results

There are a total of three fine-tuning cases considered in the experiment: supervised fine-tuning (STF), direct preference optimization (DPO) and the combination of STF and DPO respectively. To answer the second research question mentioned in Chaper 1, the original LLAMA 3.1 language model is chosen as the benchmark. From the testing set described in Section 3.2, the prompt and input fields are passed to the base and fine-tuned LLMs to infer the corresponding responses. These outputs serve as the candidate sequences while the test set's output fields act as the reference text or code in the evaluation metrics. For the BLEU metrics, the distribution of BLEU scores across different fine-tuning cases and the benchmark model are illustrated in Figure 4.3. Moreover, indicators derived from the distribution, including mean, median, variance and standard deviation, as well as the CodeBLEU scores are listed in Table 4.1.

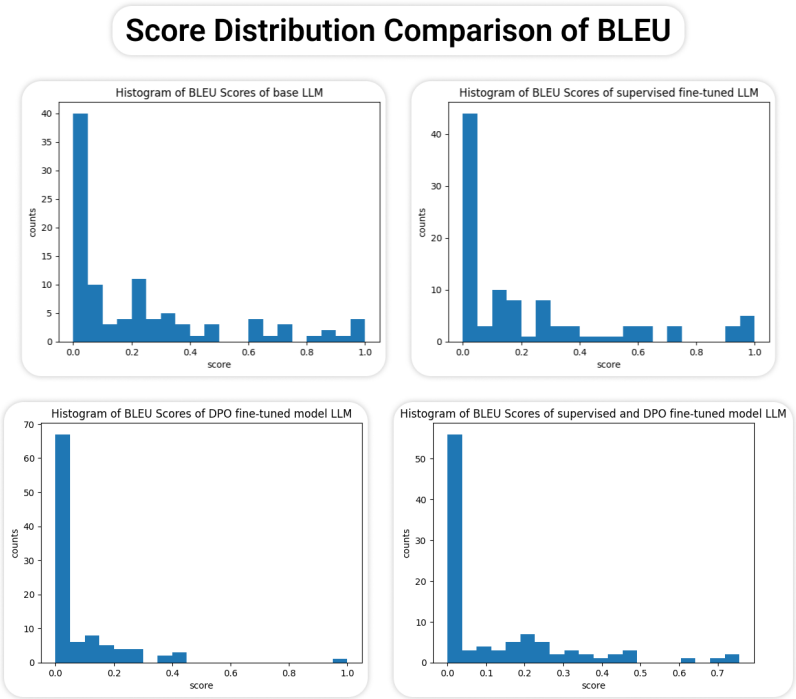

Figure 4.3. *Comparison of the base and fine-tuned LLAMA 3.1 models' BLEU score histograms.*

Table 4.1. *Properties of BLEU and CodeBLEU score distribution of base and fine-tuned LLAMA 3.1 models.*

Model types	**Score types**				
	BLEU				**CodeBLEU**
	Mean	**Median**	**Variance**	**Standard deviation**	
Base model	**0.231**	0.101	0.083	0.288	0.315
Supervised fine-tuned (STF) model	0.227	**0.117**	**0.087**	**0.295**	**0.341**
DPO fine-tuned model	0.074	0	0.021	0.144	0.228
STF and DPO fine-tuned model	0.124	0	0.033	0.181	0.253

When it comes to BERT and CodeBERT scores, the metrics yield multiple indices, consisting of recall, precision, the F_1 score and the F_3 score specific to CodeBERT. Similar to the BLEU score family, histograms in Figure 4.4, 4.5 and 4.6 illustrate the F_1 and F_3 scores from BERT and CodeBERT metrics for both base and fine-tuned models.

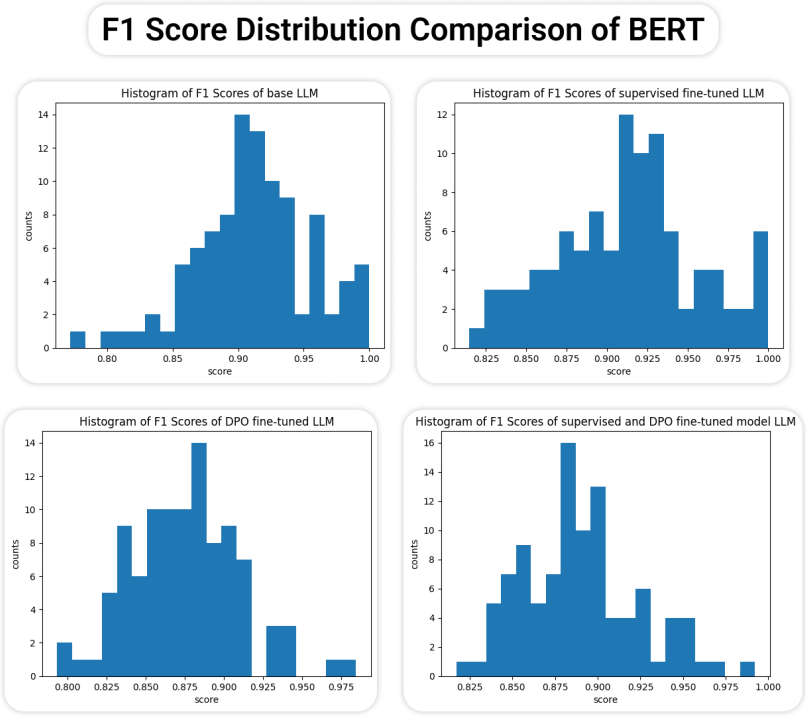

Figure 4.4. *Histograms of the F_1 scores for the BERT metric in the base and fine-tuned LLAMA 3.1 models.*

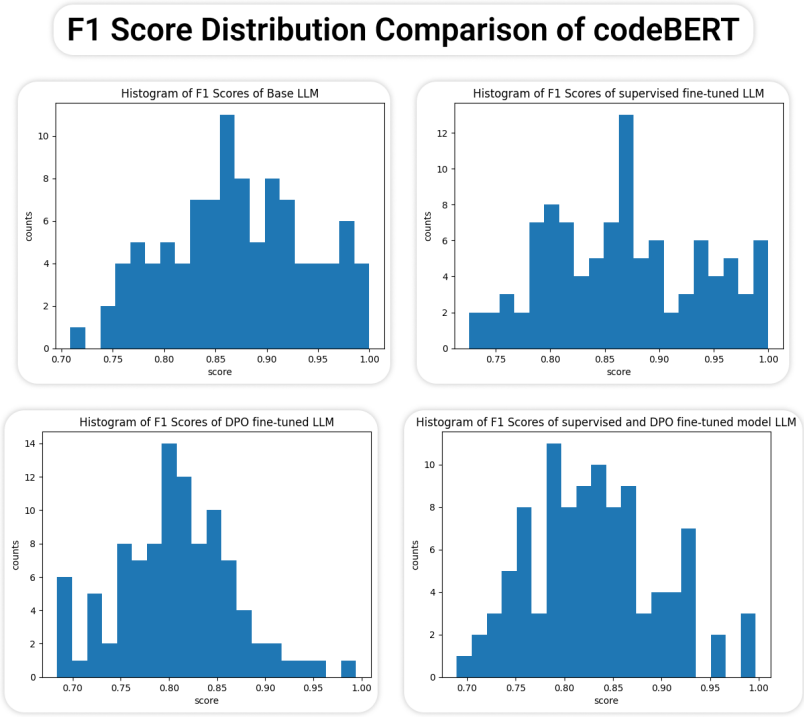

Figure 4.5. *Histograms of the F_1 scores for the CodeBERT metric in the base and fine-tuned LLAMA 3.1 models.*

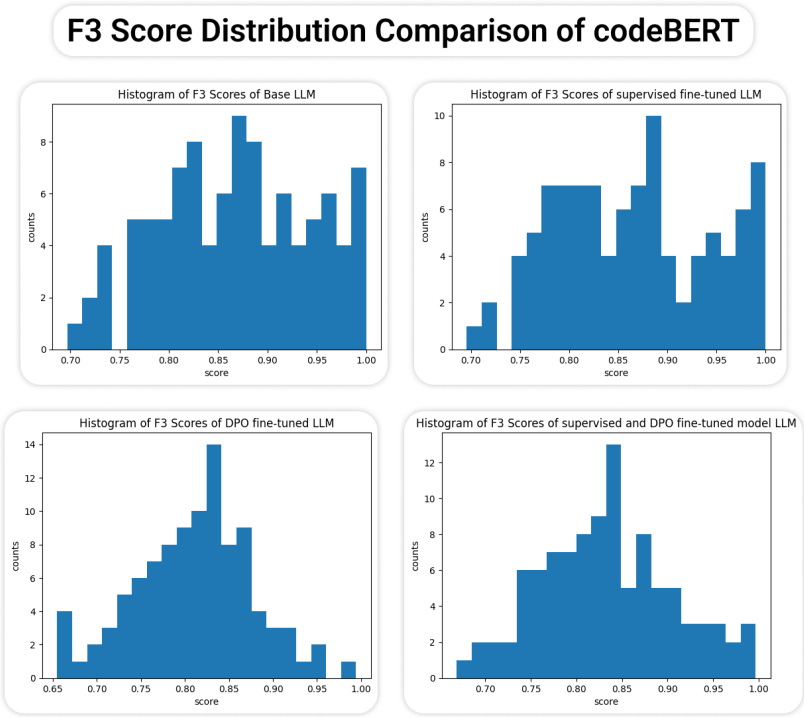

Figure 4.6. *Histograms of the F_3 scores for the CodeBERT metric in the base and fine-tuned LLAMA 3.1 models.*

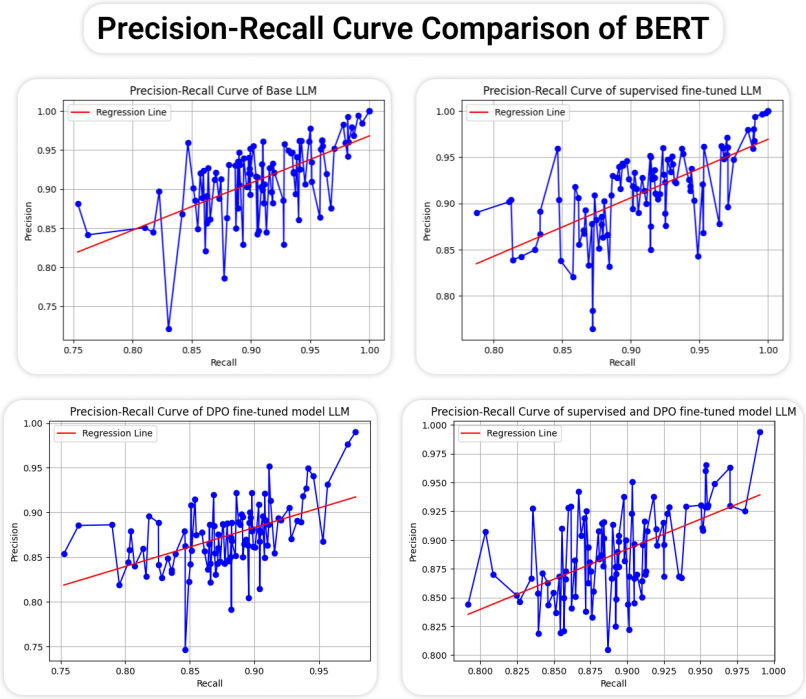

Figure 4.7. *Precision-recall curves for the BERT metric in the base and fine-tuned LLAMA 3.1 models with linearly fitted regression lines.*

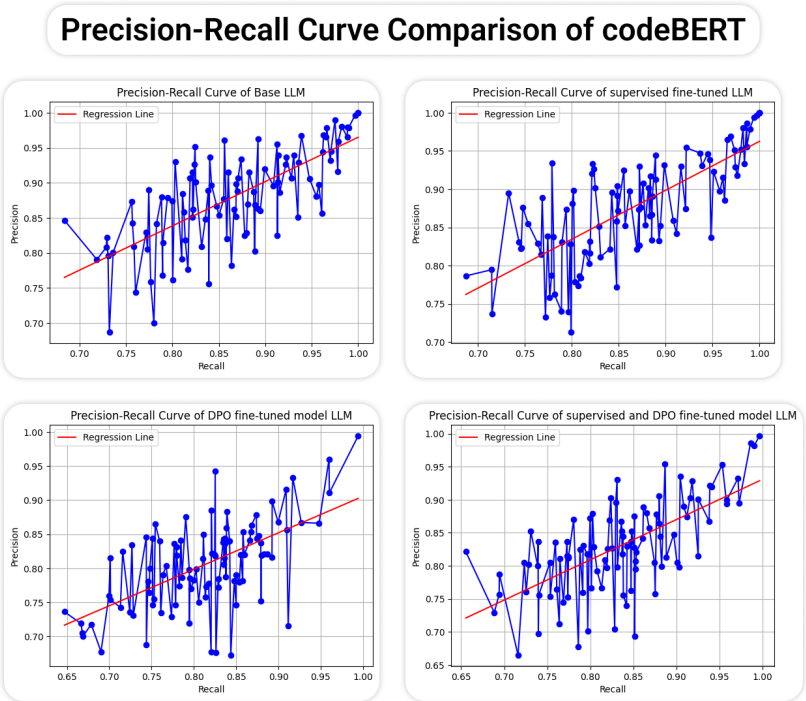

Figure 4.8. *Precision-recall curves for the CodeBERT metric in the base and fine-tuned LLAMA 3.1 models with linearly fitted regression lines.*

Table 4.2. *Properties of score distribution and regressed precision-recall curves for the BERT metric in base and fine-tuned LLAMA 3.1 models.*

Model types	**Score types**					
	BERT					
	F1				**Precision-Recall curve**	
	Mean	**Median**	**Variance**	**Standard deviation**	**Slope**	**Intercept**
Base model	**0.912**	**0.91**	**0.002**	**0.044**	**0.603**	0.365
Supervised fine-tuned (STF) model	**0.912**	**0.914**	**0.002**	**0.044**	**0.634**	0.335
DPO fine-tuned model	0.875	0.877	0.001	0.035	0.437	**0.49**
STF and DPO fine-tuned model	0.89	0.886	0.001	0.034	0.521	0.423

Table 4.3. *Properties of score distribution and regressed precision-recall curves for the CodeBERT metric in base and fine-tuned LLAMA 3.1 models.*

Model types	**Score types**									
	CodeBERT									
	F1				**F3**				**Precision-Recall curve**	
	Mean	**Median**	**Variance**	**Standard deviation**	**Mean**	**Median**	**Variance**	**Standard deviation**	**Slope**	**Intercept**
Base model	**0.872**	**0.87**	**0.005**	**0.069**	**0.867**	**0.866**	**0.006**	0.076	0.632	0.333
Supervised fine-tuned (STF) model	**0.869**	**0.869**	**0.005**	**0.071**	**0.865**	0.863	**0.006**	**0.078**	**0.64**	0.322
DPO fine-tuned model	0.808	0.806	0.004	0.06	0.812	**0.818**	0.005	0.067	0.535	**0.371**
STF and DPO fine-tuned model	0.831	0.828	0.004	0.066	0.834	0.834	0.005	0.072	0.608	0.322

Besides F_1 and F_3 scores, precision and recall scores for BERT and CodeBERT metrics across four different cases are arranged to form precision-recall curves in Figure 4.7 and 4.8. Additionally, to simplify the complex data points, we fit a linear regression line for each fine-tuning scenario and the benchmark in both BERT and CodeBERT metrics. Further details on mean, median, variance and standard deviation of the distribution along with the slopes and intercepts of the linearly fitted regression lines are shown in Table 4.2 and 4.3.

5. DISCUSSION AND ANALYSIS

As illustrated in Section 4.3, the benchmark and fine-tuned models using various techniques exhibit varying performance across the BLEU and BERT family metrics. Notably, while the benchmark slightly outperforms the supervised fine-tuned model in terms of the BLEU metric, the STF model achieves a higher CodeBLEU score than the base model. A similar trend can be seen in the F_1 and F_3 scores of BERT and CodeBERT metrics, where the benchmark model performs comparably to or slightly better than the STF model. One possible explanation is that the benchmark model has already shown competitive results on HumanEval and MBPP EvalPlus code evaluation metrics, achieving outcomes significantly higher than those of counterpart model with relatively similar amount of parameters [65]. Moreover, since the dataset utilized in the STF process is widely regarded as common, the base model has likely encountered it during the training phase. Therefore, the supervised fine-tuning may change the model's internal parameters' weights, resulting in a marginally higher CodeBLEU score. By contrast, the DPO-related fine-tuned LLMs' performance deteriorates significantly throughout both BLEU and BERT family metrics, particularly in the purely DPO fine-tuned model. This phenomenon can be attributed to the confined size of dataset used in DPO compared to STF, causing the final model's performance to be instable as noted in [41]. Another suspicion is assigning the base model's generated responses as the data for rejected field in Section 3.3 could be the cause of overfitting in fine-tuning period. Also, It is worth highlighting that the STF has contributed to mitigate the negative effects of DPO as seen in the results of the scenario where the combination of STF and DPO takes place.

The precision and recall scores from BERT and CodeBERT metrics clearly indicate a positive correlation between the two scores in both benchmark and fine-tuned LLMs. Based on the linear regression lines of precision-recall curves, the two scores show the strongest positive correlation in the STF model, closely followed by the benchmark LLM in the second place. One possible interpretation lies in the mechanisms of precision and recall calculation of BERT and CodeBERT metrics. As the candidate sequence becomes more similar to the reference one, the precision score tends to be approach the recall equivalent. A similar downward trend in the F_1 and F_3 scores for the DPO and STF-DPO models are also observed in the slopes of precision-recall curves, which could be interpreted by the previously mentioned theories.

6. CONCLUSION

The thesis approaches the fine-tuning of benchmark model, which is LLAMA 3.1 model with 8 billion parameters, using different fine-tuning techniques in various scenarios to further advance the quality of inferred code. The fine-tuning cases are structured so that the performance of the base and fine-tuned LLMs can be easily assessed and compared. To be specific, these scenarios consists of benchmark, supervised fine-tuning (STF), direct preference optimization (DPO) and the integration of STF and DPO, where a LLM-generated dataset is utilized during the DPO fine-tuning. As the fine-tuning is a resource-intensive and time-consuming task, parameter efficient fine-tuning (PEFT), including quantization methods and Low-rank adaptation (LoRA), has been applied to optimize the process throughout the research. The outcomes of the base and fine-tuned are evaluated by the BLEU and BERT metrics as well as their derivatives CodeBLEU and CodeBERT to enhance the robustness of code assessment. According the metrics, the performance differences between the benchmark and STF models are not statistically significant. Nonetheless, the decline in results is observed in fine-tuned models that incorporate the DPO method, especially in LLM fine-tuned merely with DPO. It is suspected that the use of auto-generated and relatively small dataset has caused overfitting and instability during the DPO fine-tuning. Although one is not able to make a conclusive statement indicating which fine-tuning technique yields the best improvement of generated code quality, it is evident that the dataset selection and generation substantially impact the outcome of fine-tuning.

Throughout the paper, we are able to address the earlier mentioned research questions in Chapter 1. However, we also acknowledge a number of encountered limitations during the experiment. One of them is the lack of unique and distinctive dataset so that the possibility that the model learned the dataset beforehand can be isolated. Moreover, most available datasets are tailored for the STF method, which necessitates the practice of extracting and generating a DPO-compatible dataset with the help of the benchmark model. This has led to underperformance in the examined evaluation metrics due to possible overfitting and instability. This obstacle can be overcome by applying the resourceful and high-quality dataset that is compatible with the DPO method in future studies.

REFERENCES

[1] Stiennon, N., Ouyang, L., Wu, J., Ziegler, D. M., Lowe, R., Voss, C., Radford, A., Amodei, D. and Christiano, P. *Learning to summarize from human feedback*. 2022. arXiv: 2009.01325 [cs.CL]. URL: https://arxiv.org/abs/2009.01325.

[2] Jiang, D. Analysis on the Application of Artificial Intelligence in the Medical Field. *2020 8th International Conference on Orange Technology (ICOT)*. 2020, pp. 1–4. DOI: 10.1109/ICOT51877.2020.9468742.

[3] ji, L., Wang, Z., Chen, M., Fan, S., Wang, Y. and Shen, Z. How Much Can AI Techniques Improve Surface Air Temperature Forecast? -A Report from AI Challenger 2018 Global Weather Forecast Contest. *Journal of Meteorological Research* 33 (Oct. 2019), pp. 1–4. DOI: 10.1007/s13351-019-9601-0.

[4] Xu, H., Zhao, Y., Zhao, D., Duan, Y. and Xu, X. Improvement of disastrous extreme precipitation forecasting in North China by Pangu-weather AI-driven regional WRF model. *Environmental Research Letters* 19 (May 2024). DOI: 10.1088/1748-9326/ad41f0.

[5] IBM Data and AI Team. *Types of artificial intelligence*. en. https://www.ibm.com/think/topics/artificial-intelligence-types. Accessed: 2024-9-18. Dec. 2023.

[6] OpenAI. *Introducing ChatGPT*. https://openai.com/index/chatgpt/. Accessed: 2024-9-18. Nov. 2022.

[7] Pack, A., Barrett, A. and Escalante, J. Large language models and automated essay scoring of English language learner writing: Insights into validity and reliability. *Computers and Education: Artificial Intelligence* 6 (2024), p. 100234. ISSN: 2666-920X. DOI: https://doi.org/10.1016/j.caeai.2024.100234. URL: https://www.sciencedirect.com/science/article/pii/S2666920X24000353.

[8] Wang, J. and Chen, Y. A Review on Code Generation with LLMs: Application and Evaluation. *2023 IEEE International Conference on Medical Artificial Intelligence (MedAI)*. 2023, pp. 284–289. DOI: 10.1109/MedAI59581.2023.00044.

[9] Niu, C., Zhang, T., Li, C., Luo, B. and Ng, V. *On Evaluating the Efficiency of Source Code Generated by LLMs*. 2024. arXiv: 2404.06041 [cs.SE]. URL: https://arxiv.org/abs/2404.06041.

[10] Tian, R., Ye, Y., Qin, Y., Cong, X., Lin, Y., Pan, Y., Wu, Y., Hui, H., Liu, W., Liu, Z. and Sun, M. *DebugBench: Evaluating Debugging Capability of Large Language Models*. 2024. arXiv: 2401.04621 [cs.SE]. URL: https://arxiv.org/abs/2401.04621.

[11] Brown, S. *Machine learning, explained*. en. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained. Accessed: 2024-10-20. Apr. 2021.

[12] *What is unsupervised learning?* en. https://www.ibm.com/topics/unsupervised-learning. Accessed: 2024-10-21. Aug. 2024.

[13] Index. *Unsupervised Learning: Foundations of Neural Computation*. The MIT Press, May 1999. ISBN: 9780262288033. DOI: 10.7551/mitpress/7011.003.0023. eprint: https://direct.mit.edu/book/chapter-pdf/2290120/9780262288033_cav.pdf. URL: https://doi.org/10.7551/mitpress/7011.003.0023.

[14] *What is supervised learning?* en. https://www.ibm.com/topics/supervised-learning. Accessed: 2024-10-21. Aug. 2024.

[15] Murel, J. and Kavlakoglu, E. *What is reinforcement learning?* en. https://www.ibm.com/topics/reinforcement-learning. Accessed: 2024-10-21. Aug. 2024.

[16] *Brain basics: Know your brain*. en. https://www.ninds.nih.gov/health-information/public-education/brain-basics/brain-basics-know-your-brain. Accessed: 2024-10-21.

[17] *What is a neural network?* en. https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/. Accessed: 2024-10-21. Jan. 2019.

[18] Chandra, A. L. *McCulloch-Pitts neuron — mankind's first mathematical model of A biological neuron*. en. https://towardsdatascience.com/mcculloch-pitts-model-5fdf65ac5dd1. Accessed: 2024-10-21. July 2018.

[19] pawankrgunjan Follow Improve. *What is perceptron*. en. https://www.geeksforgeeks.org/what-is-perceptron-the-simplest-artificial-neural-network/. Accessed: 2024-10-21. July 2023.

[20] Naveen. *Nomidl*. en. https://www.nomidl.com/deep-learning/difference-between-perceptron-and-neuron/. Accessed: 2024-10-21. Apr. 2022.

[21] Baheti, P. *Activation functions in neural networks [12 types & use cases]*. en. https://www.v7labs.com/blog/neural-networks-activation-functions. Accessed: 2024-10-21. July 2024.

[22] harkiran78 Follow Improve. *Artificial Neural Networks and its Applications*. en. https://www.geeksforgeeks.org/artificial-neural-networks-and-its-applications/. Accessed: 2024-10-21. June 2020.

[23] *What is a neural network?* en. https://www.ibm.com/topics/neural-networks. Accessed: 2024-10-21. Oct. 2024.

[24] *What is NLP (natural language processing)?* en. https://www.ibm.com/topics/natural-language-processing. Accessed: 2024-10-21. Oct. 2024.

[25] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L. and Polosukhin, I. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL]. URL: https://arxiv.org/abs/1706.03762.

[26] Raschka, S. *Understanding encoder and decoder LLMs*. en. https://magazine.sebastianraschka.com/p/understanding-encoder-and-decoder. Accessed: 2024-10-22. June 2023.

[27] Bai, Y. *Why are most LLMs decoder-only? - Yumo Bai*. en. https://medium.com/@yumo-bai/why-are-most-llms-decoder-only-590c903e4789. Accessed: 2024-10-22. Feb. 2024.

[28] Wydmanski, W. *Self attention vs attention in transformers*. en. https://medium.com/@wwydmanski/whats-the-difference-between-self-attention-and-attention-in-transformer-architecture-3780404382f3. Accessed: 2024-10-22. Dec. 2022.

[29] Wong, W. *What is residual connection?* en. https://towardsdatascience.com/what-is-residual-connection-efb07cab0d55. Accessed: 2024-10-22. Dec. 2021.

[30] Bharti, A. *Unraveling transformers: A deep dive into self-attention and cross-attention mechanisms*. en. https://medium.com/@abhinavbhartigoml/unraveling-transformers-a-deep-dive-into-self-attention-and-3e37dc875bea. Accessed: 2024-10-23. Feb. 2024.

[31] OpenAI. *New GPT-3 capabilities: Edit & insert*. https://openai.com/index/gpt-3-edit-insert/. Accessed: 2024-9-18. Mar. 2022.

[32] Cook, A. and Karakuş, O. LLM-Commentator: Novel fine-tuning strategies of large language models for automatic commentary generation using football event data. en. *Knowl. Based Syst.* 300.112219 (2024), p. 112219.

[33] Sonnenburg, A., Lugt, B. van der, Rehn, J., Wittkowski, P., Bech, K., Padberg, F., Eleftheriadou, D., Dobrikov, T., Bouwmeester, H., Mereu, C., Graf, F., Kneuer, C., Kramer, N. I. and Blümmel, T. Artificial intelligence-based data extraction for next generation risk assessment: Is fine-tuning of a large language model worth the effort? en. *Toxicology* 508.153933 (2024), p. 153933.

[34] Lai, Z., Wu, T., Fei, X. and Ling, Q. BERT4ST:: Fine-tuning pre-trained large language model for wind power forecasting. en. *Energy Convers. Manag.* 307.118331 (2024), p. 118331.

[35] Zhang, J., Zhang, C., Lu, J. and Zhao, Y. Domain-specific large language models for fault diagnosis of heating, ventilation, and air conditioning systems by labeled-data-supervised fine-tuning. en. *Appl. Energy* 377.124378 (2025), p. 124378.

[36] Li, R., Cao, M., Fu, D., Wei, W., Wang, D., Yuan, Z., Hu, R. and Deng, W. Deciphering language disturbances in schizophrenia: A study using fine-tuned language models. en. *Schizophr. Res.* 271 (2024), pp. 120–128.

[37] Datta, S. *Comprehensive guide to fine tuning a large language model*. en. https://blog.monsterapi.ai/blogs/fine-tune-a-large-language-model-llm-guide-2023/. Accessed: 2024-10-23. July 2023.

[38] Das, S. *Fine tune large language model (LLM) on a custom dataset with QLoRA*. en. https://dassum.medium.com/fine-tune-large-language-model-llm-on-a-custom-dataset-with-qlora-fb60abdeba07. Accessed: 2024-10-23. Jan. 2024.

[39] *Prepare & process the data*. en. https://www.datatopolicy.org/navigator/prepare-process-the-data. Accessed: 2024-10-23.

[40] Yang, H., Zhang, Y., Xu, J., Lu, H., Heng, P.-A. and Lam, W. Unveiling the generalization power of fine-tuned large language models. *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2024.

[41] Devlin, J., Chang, M.-W., Lee, K. and Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North*. Stroudsburg, PA, USA: Association for Computational Linguistics, 2019.

[42] Mahabadi, R. K., Belinkov, Y. and Henderson, J. Variational Information Bottleneck for effective low-resource fine-tuning. (2021). eprint: 2106.05469.

[43] Jin, H., Wei, W., Wang, X., Zhang, W. and Wu, Y. Rethinking learning rate tuning in the era of Large Language Models. (2023). eprint: 2309.08859.

[44] Liu, Y., He, H., Han, T., Zhang, X., Liu, M., Tian, J., Zhang, Y., Wang, J., Gao, X., Zhong, T., Pan, Y., Xu, S., Wu, Z., Liu, Z., Zhang, X., Zhang, S., Hu, X., Zhang, T., Qiang, N., Liu, T. and Ge, B. Understanding LLMs: A comprehensive overview from training to inference. (2024). eprint: 2401.02038.

[45] Parti, A. *What is reinforcement learning from human feedback (RLHF)?* en. https://pareto.ai/blog/what-is-rlhf. Accessed: 2024-9-20. Nov. 2023.

[46] Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., Laroussilhe, Q. de, Gesmundo, A., Attariyan, M. and Gelly, S. Parameter-efficient transfer learning for NLP. (2019). eprint: 1902.00751.

[47] Meta. *Introducing LLaMA: A foundational, 65-billion-parameter large language model*. https://ai.meta.com/blog/large-language-model-llama-meta-ai/. Accessed: 2024-9-21. Feb. 2023.

[48] Krishnamoorthi, R. Quantizing deep convolutional networks for efficient inference: A whitepaper. (2018). eprint: 1806.08342.

[49] CuriosityDeck. *Quantization aware training*. en. https://medium.com/@curiositydeck/quantization-aware-training-b80272380098. Accessed: 2024-9-21. Feb. 2024.

[50] Zhang, B., Liu, Z., Cherry, C. and Firat, O. When scaling meets LLM finetuning: The effect of data, model and finetuning method. (2024). eprint: 2402.17193.

[51] *unsloth/Meta-Llama-3.1-8B-bnb-4bit · Hugging Face*. https://huggingface.co/unsloth/Meta-Llama-3.1-8B-bnb-4bit. Accessed: 2024-10-23.

[52] Dettmers, T. and Zettlemoyer, L. *The case for 4-bit precision: k-bit Inference Scaling Laws*. 2023. arXiv: 2212.09720 [cs.LG]. URL: https://arxiv.org/abs/2212.09720.

[53] Xu, L., Xie, H., Qin, S.-Z. J., Tao, X. and Wang, F. L. *Parameter-Efficient Fine-Tuning Methods for Pretrained Language Models: A Critical Review and Assessment*. 2023. arXiv: 2312.12148 [cs.CL]. URL: https://arxiv.org/abs/2312.12148.

[54] Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L. and Chen, W. *LoRA: Low-Rank Adaptation of Large Language Models*. 2021. arXiv: 2106.09685 [cs.CL]. URL: https://arxiv.org/abs/2106.09685.

[55] Dettmers, T., Pagnoni, A., Holtzman, A. and Zettlemoyer, L. *QLoRA: Efficient Finetuning of Quantized LLMs*. 2023. arXiv: 2305.14314 [cs.LG]. URL: https://arxiv.org/abs/2305.14314.

[56] Wolfe, C. R. *Understanding and using supervised fine-tuning (SFT) for language models*. en. https://cameronrwolfe.substack.com/p/understanding-and-using-supervised. Accessed: 2024-10-24. Sept. 2023.

[57] Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D. and Finn, C. *Direct Preference Optimization: Your Language Model is Secretly a Reward Model*. 2024. arXiv: 2305.18290 [cs.LG]. URL: https://arxiv.org/abs/2305.18290.

[58] *Evaluating LLMs: complex scorers and evaluation frameworks*. en. https://symflower.com/en/company/blog/2024/llm-complex-scorers-evaluation-frameworks/. Accessed: 2024-10-25.

[59] Papineni, K., Roukos, S., Ward, T. and Zhu, W.-J. BLEU: a method for automatic evaluation of machine translation. *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. ACL '02. Philadelphia, Pennsylvania: Association for Computational Linguistics, 2002, pp. 311–318. DOI: 10.3115/1073083.1073135. URL: https://doi.org/10.3115/1073083.1073135.

[60] Ren, S., Guo, D., Lu, S., Zhou, L., Liu, S., Tang, D., Sundaresan, N., Zhou, M., Blanco, A. and Ma, S. *CodeBLEU: a Method for Automatic Evaluation of Code Synthesis*. 2020. arXiv: 2009.10297 [cs.SE]. URL: https://arxiv.org/abs/2009.10297.

[61] Guo, D., Ren, S., Lu, S., Feng, Z., Tang, D., Liu, S., Zhou, L., Duan, N., Svyatkovskiy, A., Fu, S., Tufano, M., Deng, S. K., Clement, C., Drain, D., Sundaresan, N., Yin, J., Jiang, D. and Zhou, M. *GraphCodeBERT: Pre-training Code Representations with Data Flow*. 2021. arXiv: 2009.08366 [cs.SE]. URL: https://arxiv.org/abs/2009.08366.

[62] Panickssery, A., Bowman, S. R. and Feng, S. *LLM Evaluators Recognize and Favor Their Own Generations*. 2024. arXiv: 2404.13076 [cs.CL]. URL: https://arxiv.org/abs/2404.13076.

[63] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q. and Artzi, Y. *BERTScore: Evaluating Text Generation with BERT*. 2020. arXiv: 1904.09675 [cs.CL]. URL: https://arxiv.org/abs/1904.09675.

[64] Zhou, S., Alon, U., Agarwal, S. and Neubig, G. *CodeBERTScore: Evaluating Code Generation with Pretrained Models of Code*. 2023. arXiv: 2302.05527 [cs.SE]. URL: https://arxiv.org/abs/2302.05527.

[65] Dubey, A. et al. *The Llama 3 Herd of Models*. 2024. arXiv: 2407.21783 [cs.AI]. URL: https://arxiv.org/abs/2407.21783.