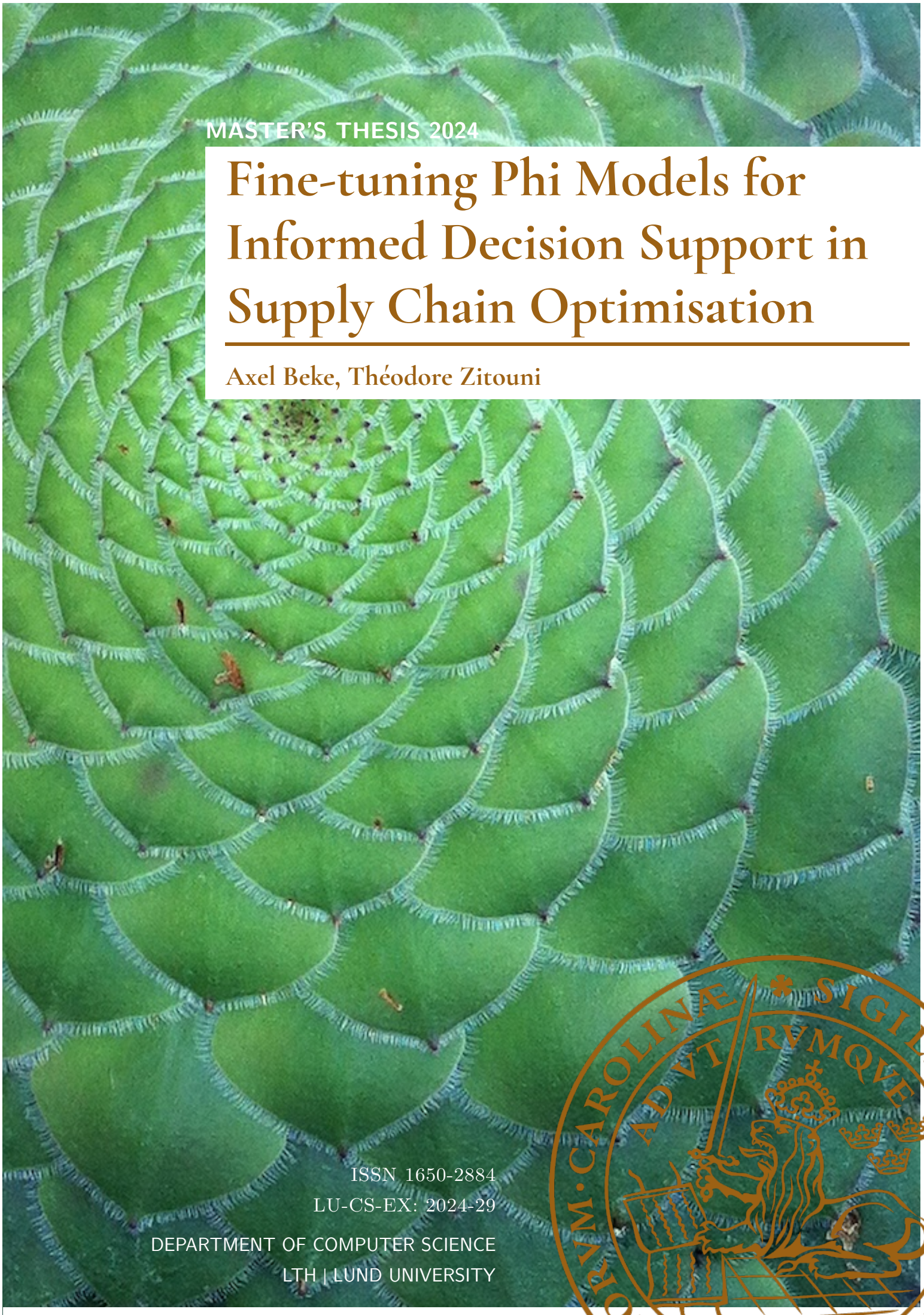


MASTER'S THESIS 2024

Fine-tuning Phi Models for Informed Decision Support in Supply Chain Optimisation

Axel Beke, Théodore Zitouni

ISSN 1650-2884
LU-CS-EX: 2024-29

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2024-29

Fine-tuning Phi Models for Informed Decision Support in Supply Chain Optimisation

Finjustering av Phi-modeller för informerat beslutsstöd inom distributionskedjeoptimering

Axel Beke, Théodore Zitouni

Fine-tuning Phi Models for Informed Decision Support in Supply Chain Optimisation

Axel Beke
ax6843be-s@student.lu.se; axel@beke.se

Théodore Zitouni
th8738zi-s@student.lu.se; tkzitouni@gmail.com

June 17, 2024

Master's thesis work carried out at Microsoft.

Supervisors: Pierre Nugues, pierre.nugues@cs.lth.se
Bahram Zarrin, bahram.zarrin@microsoft.com

Examiner: Jacek Malec, jacek.malec@cs.lth.se

Abstract

In the rapidly evolving domain of supply chain management, optimising operations for efficiency and reliability is essential. Traditional methods have struggled to keep pace with the complexity of modern logistics networks and the vast amount of data they generate. However, recent advancements in artificial intelligence offer a new avenue for innovation in this field. This research focuses on the application of Phi models, a series of small language models by Microsoft, to address two tasks in supply chain optimisation: code generation for what-if analysis and the job shop scheduling problem. We show that fine-tuning the Phi-2 model enhances its performance on these tasks, demonstrating a marked improvement in code generation capabilities with a BLEU score of 98.6%. Though it encounters challenges with the job shop scheduling problem due to its inherent complexity, it still displays a novel understanding of the problem and how to solve it, attaining a BLEU score of 21.7% using a whitespace tokenizer.

Keywords: Natural Language Processing, Small Language Models, Transformers, Fine-tuning, Phi Models, OptiGuide, Job Shop Scheduling Problem

Acknowledgements

We would like to extend our heartfelt gratitude to:

Prof. Pierre Nugues, for his continuous and tireless supervision, providing support and insights of both logistic, technical and linguistic nature.

Bahram Zarrin for the opportunity to work with state-of-the-art technology at Microsoft, providing tremendous resources and expertise.

Family and friends for their love, care and support during our five years in Lund.

Thank you.

Contents

1 Introduction **7**
- 1.1 Objective . . . 8
  - 1.1.1 Research Questions . . . 8
- 1.2 Contribution . . . 8
- 1.3 Related Work . . . 8
  - 1.3.1 Early Methods . . . 9
  - 1.3.2 Large Language Models . . . 9
  - 1.3.3 OptiGuide . . . 10

2 Theoretical Background **11**
- 2.1 Supply Chain Simulation . . . 11
  - 2.1.1 Supply Chain Management . . . 12
  - 2.1.2 Supply Chain Optimisation . . . 12
  - 2.1.3 What-if Analysis . . . 14
  - 2.1.4 Job Shop Scheduling Problem . . . 14
- 2.2 Machine Learning . . . 15
  - 2.2.1 Learning Approaches . . . 15
  - 2.2.2 Deep Learning and Neural Networks . . . 16
  - 2.2.3 Feed-forward Neural Networks . . . 16
- 2.3 Natural Language Processing . . . 18
  - 2.3.1 Numerical Representation of Text . . . 18
  - 2.3.2 The Transformer Architecture . . . 19
  - 2.3.3 Positional Encoding . . . 20
  - 2.3.4 Encoder-Decoder Architecture . . . 21
  - 2.3.5 Decoder-only Architecture . . . 22
  - 2.3.6 (Pre-)Training . . . 23
  - 2.3.7 Prompt Learning . . . 24
  - 2.3.8 Fine-tuning . . . 24
  - 2.3.9 Memory Optimisation Techniques . . . 25
  - 2.3.10 Inference . . . 26

2.3.11 Small Language Models – Phi Models . . . 26

3 Datasets **29**
3.1 Considerations . . . 29
3.2 OptiGuide Benchmark Datasets . . . 30
3.2.1 Data Samples and Statistics . . . 30
3.3 Job Shop Scheduling Dataset . . . 32
3.3.1 Prompt Learning Data . . . 33

4 Method **35**
4.1 Data Gathering and Formatting . . . 36
4.2 Base Model . . . 36
4.2.1 Baseline Prompting . . . 36
4.3 Fine-tuned Models . . . 36
4.3.1 Fine-tuning Process . . . 36
4.3.2 Infrastructure . . . 37
4.3.3 Memory Optimisation Techniques in Python . . . 39
4.3.4 Training Process . . . 40
4.4 Evaluation . . . 41
4.4.1 Metrics . . . 41
4.4.2 Evaluating the OptiGuide Benchmark Datasets . . . 43
4.4.3 Evaluating the Job Shop Scheduling Dataset . . . 44

5 Results **45**
5.1 OptiGuide Benchmark Datasets . . . 45
5.2 Job Shop Scheduling Dataset . . . 47

6 Discussion **51**
6.1 Implications of Results . . . 51
6.1.1 OptiGuide Benchmark Datasets . . . 52
6.1.2 Job Shop Scheduling Dataset . . . 53
6.2 Fine-tuning SLMs . . . 54
6.3 Considerations and Limitations . . . 55
6.3.1 Resources . . . 55
6.3.2 Ethical Considerations . . . 55

7 Conclusions **57**
7.1 Main Conclusions . . . 57
7.2 Future Work . . . 58

References **61**

Appendix A Abbreviations **69**

Appendix B A Small Note on Cover Art **71**

Appendix C Division of Work **73**

Chapter 1

Introduction

Supply chain optimisation has historically been a problem of complex nature due to its large scale, complexity in relations and hierarchies, dynamic nature and the natural uncertainty of planning. Computational advancements have improved supply chain optimisation and recently, enterprise businesses have found a research interest in leveraging natural language processing (NLP) techniques to solve tasks in the supply chain domain. The use of small language models (SLMs) such as Phi models (Gunasekar et al., 2023) can enable novel solutions to what-if scenarios and scheduling tasks, among different optimisation problems. The use of SLMs in this context is a relatively unexplored field considering the recent impact of advancements in artificial intelligence (AI) (Oliveira and Pereira, 2023; Shavaki and Ghahnavieh, 2022).

SLMs are constructed using the same foundational architectural structures employed in building large language models (LLMs). These LLMs have garnered widespread acclaim for their remarkable proficiency in processing and generating natural language, as well as excelling in tasks such as code generation and optimisation across various domains. We employ Phi-2, an SLM, to harness state-of-the-art (SOTA) NLP technologies for two distinct tasks within supply chain optimisation. Moreover, SLMs present a unique advantage in their adaptability for fine-tuning, allowing them to specialise in particular contexts or subjects. The smaller parameter size makes models like the Phi-2 ideally suited for fine-tuning on applied tasks due to the lesser resource requirements.

We employ SLMs to two distinct tasks within supply chain optimisation, namely code generation for what-if analysis and solving instances of the job shop scheduling problem (JSSP). These concepts are thoroughly introduced in Chapter 2, Theoretical Background. Our fine-tuned models obtain prominent results, excelling in code generation for what-if analysis, but show a modest understanding of the job shop scheduling problem and how to solve it.

1.1 Objective

On a high level, this master's thesis investigates the possibility of using generative AI together with operational data in data-driven supply chain optimisation. We utilise a small language model, more precisely Microsoft's Phi-2 model, to investigate if small language models can be fine-tuned to perform two distinct tasks in a supply chain context.

1.1.1 Research Questions

To fulfil the research objective and adhere to the scope of this thesis, three research questions have been defined:

1. How well can a small language model perform when fine-tuned on specific tasks in supply chain optimisation?

2. To what extent does fine-tuning a small language model outperform prompt learning for its pre-trained base model?

3. Does the small parameter size and resource efficiency of small language models impact their practical viability for fine-tuning on specific tasks?

1.2 Contribution

This master's thesis in computer science aims to contribute scientifically in the field of natural language processing through the process of fine-tuning Phi-2, a novel SLM, in an applied context. As Phi models are relatively recent and unexplored, we explore the performance in a number of settings. Using the Phi-2 model, we find it is of interest how the model performs with regard to the contextualised data and what possibilities fine-tuning presents. We consider this with regard to lesser resources required – allowing for simplified use and hosting.

Simultaneously, we aim to contribute to possible novel solutions in supply chain optimisation, leveraging the above-mentioned recent advancements in artificial intelligence. In addition to this, we also explore the capabilities of a fine-tuned Phi-2 model as a job scheduling solver and code generator for what-if analysis in supply chain optimisation.

By showing a marked improvement on both tasks using fine-tuned Phi-2 models, we build upon an existing body of knowledge. We showcase that an SLM can be fine-tuned with limited resources, enabling supply chain managers to leverage the powerful capabilities of Phi models.

1.3 Related Work

This master's thesis bridges two separate areas together, and assesses the possibilities of leveraging SLMs in a supply chain context. Therefore, our research rests on previous research,

findings and state-of-the-art results in these different fields. This specific research contribution could not be possible without the recent strides made in improving the efficiency and accuracy of both large and small language models.

On the other hand, supply chain management has been the subject of extensive research for several decades. Studies have focused on enhancing various aspects of the supply chain, from demand planning to optimisation and job scheduling. The advent of digital technologies has further revolutionised this field, with researchers exploring the potential of artificial intelligence and machine learning in enhancing supply chain processes.

The intersection of the subject areas supply chain optimisation and machine learning is relatively new, and this thesis contributes to this emerging body of research. No studies have previously been done on the potential of leveraging an SLM like Phi-2 in a supply chain context.

1.3.1 Early Methods

We introduce the early methods employed in the two fields we apply our research to.

Supply Chain Optimisation

Early methods in supply chain optimisation focused on efficiency, cost reduction, and improving coordination among various stages. One significant early method is the Just-in-Time (JIT) inventory system, developed by Toyota in the 1970s (Ohno, 1988). JIT focuses on reducing waste and improving efficiency by synchronising production with demand, thereby minimising inventory levels and associated costs (Ohno, 1988). Early methods, including JIT, in supply chain optimisation laid the groundwork for modern, more technological approaches by emphasising efficiency and cost reduction. More novel approaches use modern technology such as linear programming (LP) and large amounts of data.

Job Shop Scheduling

Early methods used to solve JSSPs consist mostly of manual solutions. Later, constraint programming solvers, such as Google's OR-Tools, were released. Constraint programming solvers always strive to find an optimal solution. However, due to the complexity that easily arises in these tasks, some solutions found are not optimal.

1.3.2 Large Language Models

The evolvement of large language models has witnessed remarkable advancements over the past decade, driven by breakthroughs in NLP and deep learning techniques. A major breakthrough was made when Vaswani et al. (2017) introduced the Transformer. The widespread adoption of Transformer based architectures revolutionised NLP by offering superior performance in tasks such as language modelling, machine translation and text generation. Transformers, with their self-attention mechanisms, capture long-range dependencies in text more effectively than previous architectures.

Following the Transformer, the advent of Generative Pre-trained Transformers (GPT) marked a significant milestone in the development of large language models (Brown et al., 2020). Introduced by OpenAI, GPT demonstrates the effectiveness of pre-training large-scale Transformer based architectures on vast text corpora, enabling the generation of coherent and relevant text (Brown et al., 2020).

Contrary to LLMs continuously growing in parameter size and complexity, smaller language models have emerged in recent years, offering impressive reasoning capabilities while staying slim in parameter size. This development has lowered the resource requirements for fine-tuning language models to perform specific tasks.

1.3.3 OptiGuide

Along with the rise of large language models in the beginning of this decade, novel possibilities have emerged in the domain of supply chain optimisation. Early research in this field has been conducted through OptiGuide (Li et al., 2023a), where the possibility of using large language models in supply chain management was explored. In the paper, supply chains are modelled as mixed integer program (MIP) problems. Such solutions can be mathematically optimised by SOTA optimisers such as Gorubi. Li et al. (2023a) developed a holistic LLM framework, OptiGuide. Furthermore, they continue by incorporating GPT-4 (Brown et al., 2020) for reasoning and answering of what-if questions, an integer programming solver/optimiser for MIP and databases for information specific queries. As the Microsoft researchers behind this paper state, the OptiGuide framework serves as a cornerstone for future research (Li et al., 2023a). For all intents and purposes, this Microsoft Research paper provides an excellent starting point for our own thesis research.

More specifically, Li et al. (2023a) themselves state that one interesting future direction is the use of smaller models, in contrast to the massive GPT-4. As an alternative to GPT models Li et al. (2023a) specifically reference the Phi models that are used in this thesis research, stating that smaller models allow for more affordable hosting and fine-tuning. In relation to this, they continue to explain that there lies a research interest in examining whether fine-tuning can help with interpreting unseen questions.

Furthermore, OptiGuide uses a number of benchmark datasets that are provided and can be found in Microsoft's open source GitHub repositories. Li et al. (2023a) state that the methodology and the benchmark datasets are to be seen as stand-alone contributions that can be used for future evaluation. The benchmark datasets are further explained in Chapter 3 of this thesis.

Chapter 2
Theoretical Background

This chapter provides rigorous background, firstly through an introduction to two areas in supply chain optimisation and secondly through an introduction to the NLP technologies covered in this thesis. Section 2.1 dives deep into how supply chains can be mathematically modelled, simulated and optimised. Moreover, Section 2.1.4 explains what the job shop scheduling problem is.

The second part of this chapter is a deep-dive into the evolution of neural networks and language technology techniques. Section 2.2 introduces modern machine learning and neural networks. Section 2.3 describes how modern natural language processing stems from early ideas of numerical representation of natural language as well as modern techniques used in this field. The landscape of text representation and interpretation has undergone significant transformations in recent decades. The development is driven by advancements in computing and the emergence of sophisticated artificial intelligence techniques such as deep learning and neural networks, outlined in Section 2.2.2. These techniques, including machine learning and deep learning, leverage intricate architectures to effectively capture the nuances of natural language, covering tone, intent and structure with remarkable precision (Nugues, 2016).

2.1 Supply Chain Simulation

Supply chains are a critical component of any global business enterprise or organisation, covering the intricate network of processes involved in bringing goods and services from their point of origin to the end consumer. It involves the coordination of resources, information and activities across multiple stages, from raw material extraction to production, distribution and ultimately, consumption. At its core, a supply chain comprises various entities such as suppliers, manufacturers, retailers, logistics providers and end users.

The simulation of supply chains is vital in order to model and analyse the dynamics of supply chain processes, enabling enterprise businesses to make informed decisions and optimise their operations. By simulating various scenarios, such as changes in demand, disruptions in supply, or alterations in production schedules – so called what-if scenarios – companies can identify bottlenecks, inefficiencies and vulnerabilities within their supply chains. Optimising supply chain simulation ensures that businesses can proactively address challenges, improve resilience, minimise costs and enhance overall performance.

2.1.1 Supply Chain Management

Organisations devote major resources to ensure supply chains are optimised, robust and explainable. Supply chain management plays a pivotal role in enhancing operational efficiency, reducing costs and tackling unexpected issues. By optimising the flow of materials, information and funds across the supply chain, organisations can minimise inventory holding costs, streamline production processes and respond rapidly to changes in parameters. Furthermore, an efficient supply chain enables businesses to mitigate risks associated with disruptions such as unforeseen demand fluctuations.

Supply chain management serves as the backbone of modern commercialism, facilitating the flow of goods and services across the global marketplace. Its importance lies in its ability to drive operational efficiency, reduce costs, mitigate risks, promote sustainability and adapt to the dynamic demands of the market. As businesses continue to evolve in a rapidly changing landscape, an effective supply chain management strategy remains important.

2.1.2 Supply Chain Optimisation

In order to maximise the efficiency and effectiveness of the entire supply chain network, organisations optimise their supply chains. Supply chain optimisation involves strategic planning, decision-making and operational execution to achieve the best possible outcomes in terms of cost, quality, speed and flexibility. This process utilises mathematical modelling and algorithms to solve complex problems related to the efficient allocation of resources, minimisation of costs and maximisation of performance metrics.

One widely used mathematical technique is linear programming, which formulates supply chain optimisation problems as a set of linear equations and inequalities, with the objective of maximising or minimising a linear objective function subject to various constraints. Additionally, integer programming and mixed integer programming techniques are utilised when decision variables are required to be integers, often applicable in scenarios such as production quantities or facility locations. One example of software that optimises supply chains utilising these techniques is Gurobi. By modelling supply chains in this manner, they can easily be transcribed to Python code. Utilising programming, models can be altered and optimised to facilitate what-if analysis.

An infamously NP-hard optimisation problem is the Travelling Salesman problem (TSP), commonly seen in the context of algorithms in computer science. It aims to find the shortest route for a salesman to visit all the cities in a given set of cities and return to the starting

point. There are several known formulations of this classic problem. The TSP can be formulated according to the Miller-Tucker-Zemlin formulation (Miller et al., 1960). Below, in Equation 2.1, we have an optimisation problem that is comprised of the objective, minimising $\sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij}$ where c_{ij} denotes the distance between cities i and j, x_{ij} is a binary variable denoting whether the salesman travels directly between city i and j or not. Furthermore, minimising the objective is subject to the constraints that each city in the given set must be visited once, $\sum_{j=1, j \neq i}^{n} x_{ij} = 1$, and the return to the starting point. The last two constraints ensure that the optimised solution does not contain any sub-cycles, commonly known as the sub-tour elimination constraints in the Miller-Tucker-Zemlin formulation (Miller et al., 1960). The dummy variable u_i is used to understand in which order cities have been visited.

$$
\begin{aligned}
\text{Minimise} \quad & \sum_{i=1}^{n} \sum_{j \neq i, j=1}^{n} c_{ij} x_{ij} & & \\
\text{s.t.} \quad & x_{ij} \in \{0, 1\} & & i, j = 1, \ldots, n; \\
& \sum_{i=1, i \neq j}^{n} x_{ij} = 1 & & j = 1, \ldots, n; \\
& \sum_{j=1, j \neq i}^{n} x_{ij} = 1 & & i = 1, \ldots, n; \\
& u_i - u_j + 1 \leq (n-1)(1 - x_{ij}) & & 2 \leq i \neq j \leq n; \\
& 2 \leq u_i \leq n & & 2 \leq i \leq n.
\end{aligned} \tag{2.1}
$$

In the same manner as the TSP, a simple, general example of a supply chain can be formulated. Let's envision an enterprise offering two different products: A and B. These two products are manufactured in one of two factories. Products A and B are manufactured from the same natural material m. The material m is sourced from three distinct suppliers. The finished products A and B are then distributed to three separate retailers. The primary objective is to minimise overall costs while ensuring the demand is met satisfactorily. These costs encompass the expenses incurred in acquiring the material m, the processing expenses associated with manufacturing A and B, and the shipping costs to the retail grocery stores. This example follows, for simplicity and explainability, the example of Li et al. (2023a).

The optimisation problem is framed as a mixed integer program, where $x_{s,f}$ denotes the quantity of material purchased from supplier s for factory f. We define $y^A_{f,r}$ and $y^B_{f,r}$ as the amount of products A and B, respectively, delivered from factory f to retailer r. Each supplier has a capacity C_s, and each retailer r has a demand D^A_r and D^B_r for products A and B respectively. The cost $c_{s,f}$ is incurred for each unit of material m purchased from supplier s for each factory f. Furthermore, there are manufacturing costs g^A_f and g^B_f associated with each unit of each product in factory f. Lastly, a shipping cost $h_{f,r}$ is applied for each unit of product delivered from factory f to retailer r. Both this problem and the TSP can be transcribed to Python code, facilitating code generation for possible scenarios in what-if analysis. Similarly to the TSP modelled with the Miller-Tucker-Zemlin formulation, the optimisation problem is as

follows

$$
\begin{aligned}
\text{Minimise} \quad & \sum_{s,f} x_{s,f} \cdot c_{s,f} + \sum_{f,r} y^A_{f,r} \cdot h^A_f \quad + \\
& \sum_{f,r} y^B_{f,r} \cdot h^B_f + \sum g_{f,r} \cdot (y^A_{f,r} + y^B_{f,r}) \\
\text{s.t.} \quad & \sum_{f} x_{s,f} \leq C_s && \forall s; \\
& \sum_{s} x_{s,f} = \sum_{r} (y^A_{f,r} + y^B_{f,r}) && \forall f; \\
& \sum_{f} y^A_{f,r} \geq D^A_r && \forall r; \quad (2.2) \\
& \sum_{f} y^B_{f,r} \geq D^B_r && \forall r; \\
& x_{s,f}, y^A_{f,r}, y^B_{f,r} \in \mathbb{Z}^+ && \forall s, f, r.
\end{aligned}
$$

2.1.3 What-if Analysis

The practice of conducting what-if scenarios in supply chain optimisation entails simulating various hypothetical situations to gauge their potential impact on the performance and resilience of the supply chain. These scenarios are important for both supply chain managers and engineers, enabling them to proactively anticipate and address risks while optimising resource allocation.

For instance, companies may simulate abrupt shifts in customer demand, disruptions in transportation or production, fluctuations in commodity prices or other events. By adjusting different parameters and assumptions within supply chain models, supply chain managers can evaluate how these scenarios might influence critical performance metrics.

Importantly, this analytical approach extends beyond supply chain management, encompassing a wide array of scheduling, logistics and optimisation challenges. These problems can be characterised by variables such as inventory levels, constraints and demand patterns (Watson et al., 2012).

2.1.4 Job Shop Scheduling Problem

The job shop scheduling problem is an optimisation challenge classified as NP-hard, focusing on the allocation of jobs to machines in a way that optimises a certain objective, typically minimising the makespan – total processing time. It involves scheduling multiple jobs, $j_1 \dots j_n$, each consisting of a sequence of tasks, $t_{j,1} \dots t_{j,n}$, which must be processed on specific machines, $m_0 \dots m_m$ for a set duration (Taillard, 1993). This process is depicted in Figure 2.1.

The complexity of the problem arises from several constraints: each machine can only handle one task at a time, tasks must follow a specific order and tasks must be run on a specific machine (Taillard, 1993). This leads to complicated solutions as the problem increases in size. Solutions to JSSP are critical in manufacturing and production environments where efficient scheduling can lead to significant time and cost savings. The problem is a fundamental study in the field of operations research and industrial engineering, with ongoing research exploring various algorithms and methods to find effective solutions.

Tasks (machine_id, time units)

Job 1	(0, 3)	(1, 2)	(2, 2)
Job 2	(0, 2)	(2, 1)	(1, 4)
Job 3	(1, 4)	(0, 2)	(2, 3)

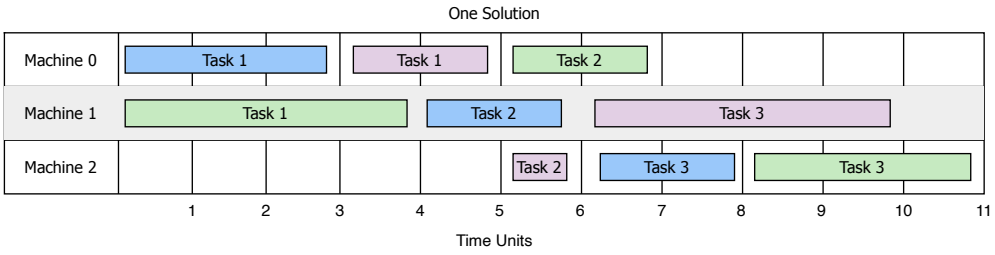


Figure 2.1: An example of the job shop scheduling problem. The upper table shows three jobs with three tasks each. Each task shows what machine it needs to be run on and for how long, respectively. The lower table shows one possible solution to the problem.

2.2 Machine Learning

Machine learning, a subset of artificial intelligence, is a method in statistical learning that has proven successful in more recent advancements of artificial intelligence. Machine learning enables computers – machines – to learn from data through generalisation, pattern recognition and inference (Goodfellow et al., 2016). In order to achieve learning, models are built and trained using data, through either supervised or unsupervised learning, explained further in Section 2.2.1. The term machine learning was coined by IBM researcher Arthur Samuel already back in 1959 as "the field of study that gives computers the ability to learn without being explicitly programmed" (Russell and Norvig, 2020).

2.2.1 Learning Approaches

In the domain of machine learning, there exist different learning approaches based on the given task and data.

Supervised Learning

Supervised learning can be applied when using labelled data. The model learns from data that is organised in pairs of input-output to find the appropriate mapping (Russell and Norvig, 2020). In the context of natural language processing, such an input-output pair could typically be a sequence of text to be labelled with a language.

Unsupervised Learning

Unsupervised learning, on the other hand, operates without labelled data. Instead of relying on input-output pairs, unsupervised learning algorithms autonomously explore data to uncover hidden patterns or structures (Russell and Norvig, 2020). In natural language processing, for example, unsupervised learning is used to analyse text corpora to identify themes or cluster similar documents (Nugues, 2016). This approach enables the discovery of valuable information that may not be readily apparent through labelling. A form of unsupervised learning is masked language modelling (MLM), which makes use of masked tokens in the input sequence (Devlin et al., 2019).

2.2.2 Deep Learning and Neural Networks

As a subset of machine learning, deep learning is the approach of solving problems and tasks that for humans seem intuitive, but are hard to describe formally, by mimicking the structure of the human brain (Goodfellow et al., 2016). The term deep learning stems from the fact that the graphical display of modern neural networks involves many different layers, combined to a deep model (Goodfellow et al., 2016). The goal of all neural networks is to approximate some function f^* which, for example, maps an input x to an output y as

$$y = f^*(x) \tag{2.3}$$

where a feed-forward neural network tries to approximate $f^*(x)$ as

$$y = f(x; \theta) \tag{2.4}$$

where θ are the parameters to be learned (Goodfellow et al., 2016).

2.2.3 Feed-forward Neural Networks

The simplest deep learning architecture is the multilayer perceptron (MLP), a feed-forward neural network (Goodfellow et al., 2016). The simplest MLP is a single layer perceptron with inputs $x = x_1, x_2, \ldots, x_n$ and weights $w = w_1, w_2, \ldots, w_n$ which outputs $\hat{y}$ as

$$\hat{y} = \phi\left(\sum_{n=1}^{N} w_n x_n + b\right) \tag{2.5}$$

where b is the bias term and ϕ is the activation function, which transforms the linear combination of the weighted inputs and the bias into a non-linear output in order to capture

complex relationships (Bläckberg, 2024; Goodfellow et al., 2016). In the context of NLP (see Section 2.3) a popular activation function is the rectified linear unit (ReLU)

$$\text{ReLU}(x) = \max(0, x) \tag{2.6}$$

where x is the input to the neural network (Nugues, 2016). Note that the single layer perceptron is essentially a logistic regression, given that the activation function is the identity function.

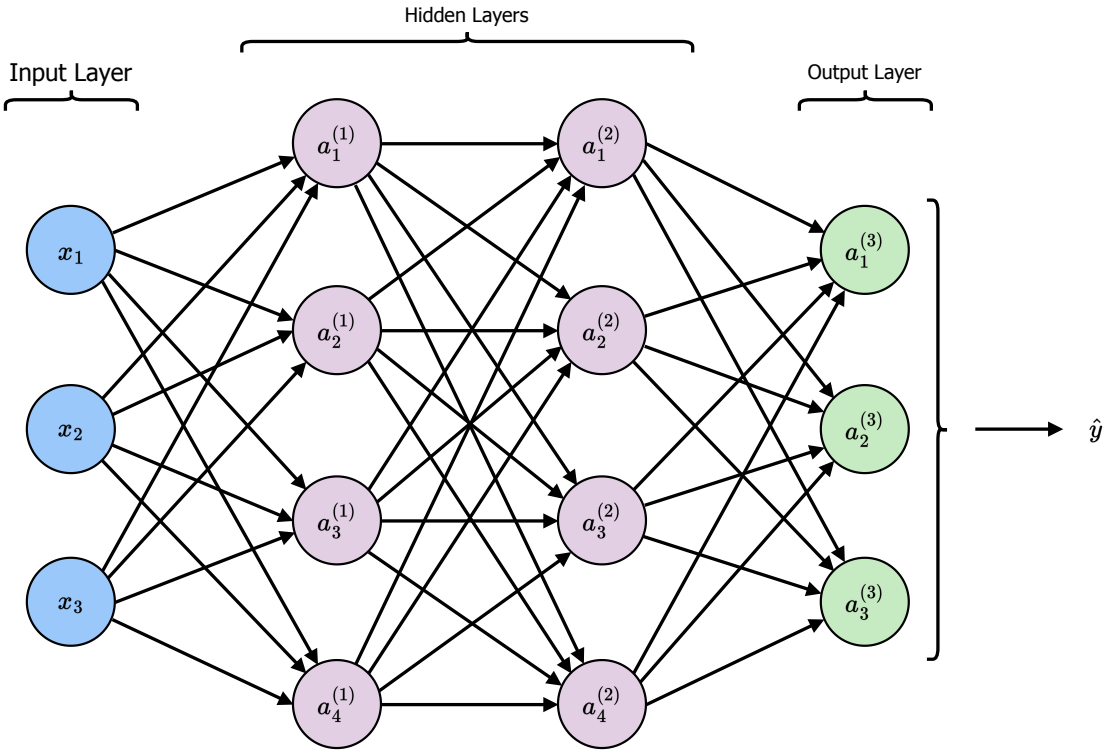


Figure 2.2: A fully connected neural network with two hidden layers and an output layer, where x_n denotes inputs, $a_n^{(i)}$ denotes nodes and $\hat{y}$ denotes the predicted output.

More advanced feed-forward neural networks have more layers, consisting of several nodes, as depicted in Figure 2.2. In each layer, each node receives the full information from each node in the previous layer (Goodfellow et al., 2016). Each node in a given layer works in parallel with the other nodes to process the received information and send it forward to the nodes in the next layer (Goodfellow et al., 2016). Typically, a modern deep feed-forward neural network consists of an input layer, several hidden layers and an output layer. During training, a deep feed-forward neural network learns by minimising the residual between its predicted outputs $\hat{y}$ and the actual labels y in the training data (Russell and Norvig, 2020). The residual, or the error, is typically minimised through a loss function such as mean squared error or binary cross entropy (Goodfellow et al., 2016). This process, known as backpropagation, computes the gradients of a predefined loss function with respect to the network's parameters (Goodfellow et al., 2016). These gradients guide the update of the parameters via optimisation algorithms like stochastic gradient descent (SGD) (Goodfellow et al., 2016; Nugues, 2016).

2.3 Natural Language Processing

The following sections provide a full explanation of all natural language processing techniques covered in this paper. It starts off in Section 2.3.1 with a justification of word embeddings, followed by an in-depth review of Transformers in Section 2.3.2, how they are trained in Section 2.3.6 and ultimately finishes off with a description of SLMs in Section 2.3.11.

In recent years, the field of natural language processing has witnessed remarkable breakthroughs, driven by the convergence of computational power, abundant data availability and innovative algorithmic approaches. One of the most notable advancements is the advent of transformer-based models, exemplified by architectures like BERT (Bidirectional Encoder Representations from Transformers) and GPT (Generative Pre-trained Transformer) (Devlin et al., 2019; Brown et al., 2020). These models have revolutionised various NLP tasks, including language translation, sentiment analysis and question answering, by leveraging self-attention mechanisms to capture contextual information effectively across long sequences of text. Additionally, techniques such as fine-tuning have enabled the adaptation of pre-trained models to specific domains or tasks with minimal additional training data, significantly reducing the time and resources required for model development. Recently, small language models such as Phi models (Gunasekar et al., 2023) have garnered increasing interest for fine-tuning purposes. Lower model complexity offers computational advantages, allowing for faster training and inference, particularly in resource-constrained environments. This trend towards smaller models underscores the importance of efficient model design and optimisation techniques in the pursuit of scalable and accessible NLP solutions.

2.3.1 Numerical Representation of Text

When dealing with NLP and language models it is important to understand that these models cannot read words and sentences the same way we humans do and instead need to rely on numerical representations of the text (Russell and Norvig, 2020). There are many ways of mapping text to a mathematical representation, and to accustom the reader to the concept, we hereby explain a simple encoding scheme named one-hot vector encoding (Russell and Norvig, 2020). Consider this set of words; {*dog*, *cat*, *bird*} and let all words be represented by a boolean vector of dimension three – because there are three words, in which exactly one of the indices are true. The resulting vectors will be: *dog* – [1,0,0], *cat* – [0,1,0], *bird* – [0,0,1]. However, if the vocabulary increases then the dimension of the word vectors increases correspondingly, which might result in very large and sparse vectors (Nugues, 2016).

Word Embeddings

One-hot vector encoding appears to be a simple and efficient method, but unfortunately, its simplicity is also its downfall. Additionally, it does not encapsulate any contextual information between different words, and therefore a more modern approach of encoding is used, often referred to as word embeddings (Pennington et al., 2014). In this method, the words are mapped to real-valued, low-dimensional dense vectors (Pennington et al., 2014). These vectors are constructed in a way that preserves the semantic relations between words, by having words that are related close to each other in the vector space (Pennington et al., 2014). The

words *happy* and *joyful* will, for example, be very close to each other in the vector space, but *happy* and *sad*, which are antonyms, tend to have opposite vectors if the embeddings are well constructed (Pennington et al., 2014). These relationships between the vectors show how the semantic meaning between words are preserved and gives an understanding of how a model might benefit from this while learning (Pennington et al., 2014).

There exist many ways of creating word embeddings, e.g. word2vec (Mikolov et al., 2013) and GloVe (Pennington et al., 2014). Traditionally, these are constructed with the use of neural network-like models. However, since the commencement of more powerful language models, e.g. Transformers (Vaswani et al., 2017), the creation of these embeddings has shifted towards newer methods (Devlin et al., 2019).

2.3.2 The Transformer Architecture

Here we provide a simplified yet in-depth overview of the Transformer architecture and its components. For a full explanation of it we refer the reader to the original paper by Vaswani et al. (2017).

The Transformer is a neural network architecture that revolutionised NLP. Unlike more traditional architectures which depend on recurrence such as recurrent neural networks (RNNs), the Transformer operates solely on attention. Moreover, this attention-mechanism enables the model to focus its attention on specific words that are more relevant than others, making it highly efficient.

Self-Attention

Self-Attention, as proposed by Vaswani et al. (2017), is the most important part of the Transformer. The idea of self-attention relies on the fact that for any sequence, some words or tokens matter more than others for a given context, and should therefore weigh in more in the prediction of the following token. For example, in the sentence

> The animal didn't cross the street because it was too tired,

the word *it* is ambiguous. It can be associated to both *The animal* and *the street*. However, self-attention allows the model to associate *it* more strongly with *The animal* than *the street*. Furthermore, this attention-mechanism is often used in parallel with other self-attention mechanisms giving rise to the multi-head attention mechanism. The idea is that each layer of self-attention will attend to different parts of the input, increasing parallelisation.

Vaswani et al. (2017) proposed this to be done using a scaled dot-product attention mechanism where a query matrix Q is multiplied by the transpose of a key matrix K, and then normalised by the dimension of d_k. The result is then sent through a softmax function and lastly multiplied by a value matrix V yielding the following:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.7}$$

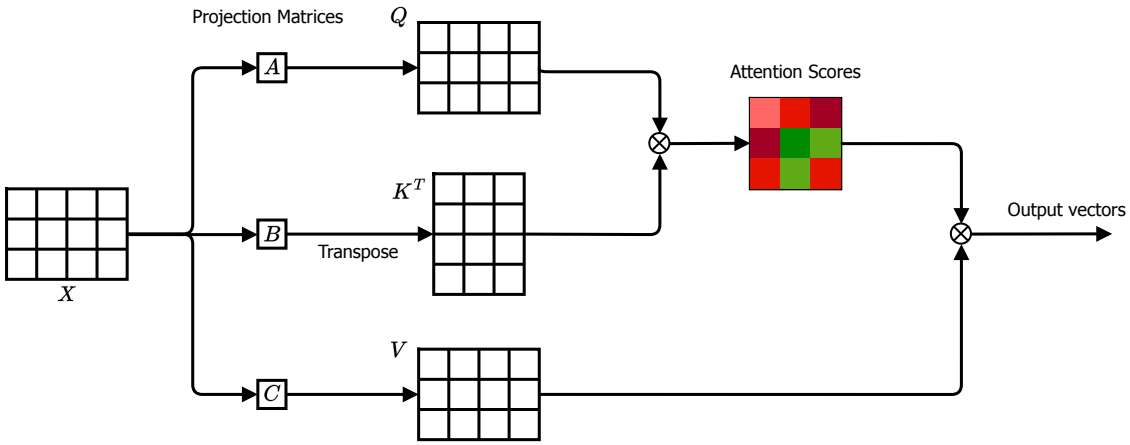


Figure 2.3: Diagram of the attention mechanism as described in Section 2.3.2

To produce these matrices, as shown in Figure 2.3, Vaswani et al. (2017) propose that each input vector (embedded word) is transformed by a learnable projection query matrix $\boldsymbol{A}$, Key matrix $\boldsymbol{B}$ and value matrix $\boldsymbol{C}$ into a query vector, key vector and value vector. All vectors for each type are then grouped together in order to create $\boldsymbol{Q}$, $\boldsymbol{K}$ and $\boldsymbol{V}$. The learnable projection matrices are parameters optimised during training by the model in order to achieve good attention. To clarify Equation 2.7, $\boldsymbol{Q}$ is multiplied by $\boldsymbol{K}^T$ acquiring attention scores between each query and key. These are then scaled down to mitigate the problem of vanishing gradients during training. The resulting vectors are passed through a softmax layer, obtaining the attention weights. These weights are then multiplied by the value matrix, computing a weighted sum of the value vectors, hence obtaining a new vector for each word that determines its context in the text. This final vector is constructed by combining information from all other words in the text, weighted by their relevance.

2.3.3 Positional Encoding

Since the model contains no recurrence, it is necessary to inject some information about the positions of the words into the embeddings. Vaswani et al. (2017) proposed a sinusoidal encoding as depicted in Figure 2.4. The sinusoidal positional encoding is computed as

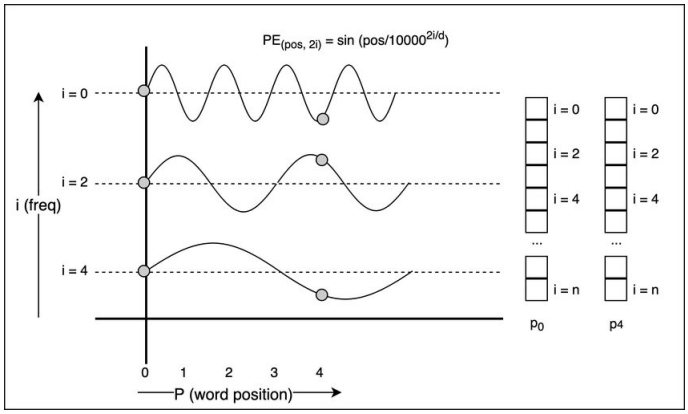


Figure 2.4: Positional encoding as illustrated by Haque and Ghani (2022).

$$\mathrm{PE}_{(pos,2i)} = \sin(pos/10000^{2i/d_{\mathrm{model}}}) \tag{2.8}$$

where pos is the position and i is the dimension, corresponding to one sinusoidal wave. The positional encoding retains the same dimensionality as the embeddings, enabling a summation between them (Vaswani et al., 2017; Haque and Ghani, 2022).

2.3.4 Encoder-Decoder Architecture

The original architecture consists of an encoder and a decoder parts as displayed in Figure 2.5. This architecture is predominantly used in sequence-to-sequence tasks such as language translation. The strength of the encoder-decoder architecture lies in its ability to transform input sequences into a latent space representation and decode them back into output sequences (Vaswani et al., 2017).

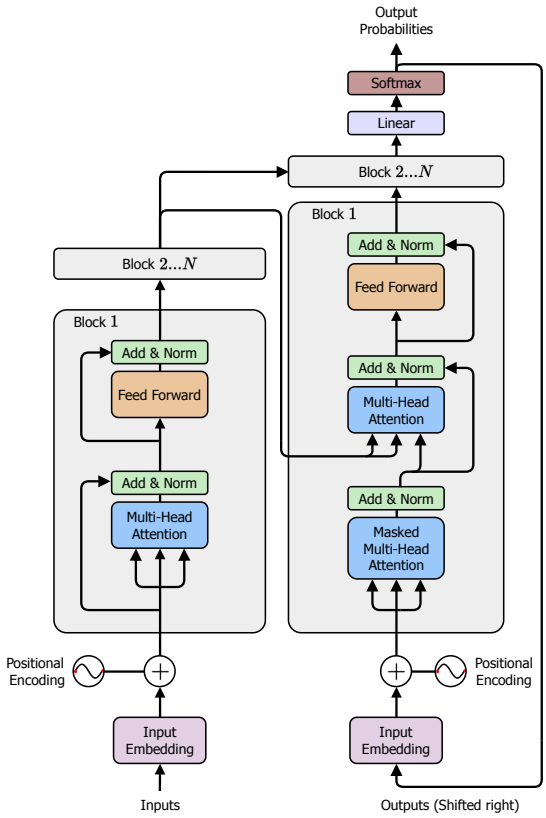


Figure 2.5: The Transformer architecture, showing the encoder (left), and decoder (right) as in the original paper by Vaswani et al. (2017) The blocks are repeated N times.

Encoder

As illustrated in Figure 2.5, the encoder is composed of N identical layers, where N is typically determined by the complexity of the task. Each layer is composed of two sub-layers; a multi-head attention layer and a fully connected neural network. Vaswani et al. (2017) proposed a two layer network with ReLU activation in between the layers.

As mentioned, the sole purpose of the encoder is to transform the input sequence into a latent space representation. This latent space representation functions as a compressed form of the original input sequence. Despite its reduced dimensionality, it is designed to maintain the critical information needed for the decoder to generate an accurate output sequence. Since the final vector outputted by the encoder is *contextualised* by the attention depending on the other words in the sentence, it is sometimes also referred to as a context vector (Vaswani et al., 2017).

Decoder

Similar to the encoder, the decoder is also composed of N identical layers. Vaswani et al. (2017) explain that each layer in the decoder contains three sub-layers. The first sub-layer performs what is known as masked attention on all previously generated tokens (Devlin et al., 2019). This means that each word can only attend to preceding words, preventing future words being used in the prediction of the next token. The second sub-layer is a regular multi-head attention layer, which takes the output of the encoder as well as the output from the first sub-layer. It uses encoder outputs as queries and keys, and the output from the first sub-layer as values, granting the ability to attend over the entire input sequence as well as the previously generated tokens (Vaswani et al., 2017). The final layer is identical to the final layer of the encoder. The final output is then sent through a final neural network layer, ultimately being soft-maxed which yields the final prediction (Vaswani et al., 2017).

As stated, the primary role of the decoder is to transform the context vector from the encoder into the output sequence. It performs this by generating one token at a time, using the context vector and all previously generated tokens (Vaswani et al., 2017). The decoder therefore plays a crucial role in ensuring that the output sequence accurately reflects the information contained in the input sequence.

The remarkable aspect of this architecture is that the encoder and decoder are trained together, allowing them to optimise their transforming skills for the task at hand. This joint training enables the model to learn the most effective ways to compress and decompress the sequence information (Vaswani et al., 2017).

2.3.5 Decoder-only Architecture

The original Transformer architecture has proven to be incredibly skillful in sequence-to-sequence tasks, such as language translation. However, using only the decoder has also proven to be very proficient. While still retaining the power and efficiency of the original Transformer, Radford et al. (2018) therefore propose the decoder-only Transformer. The decoder-only Transformer architecture utilises only the decoder part of the original Transformer by Vaswani et al. (2017), enabling it to generate new tokens based only on an input prompt. The architecture is shown in Figure 2.6. As previously noted, the main modification of the decoder-only Transformer is its lack of an encoder. However, since no encoder is used, the decoder therefore does not need the sub-layer connected to it. Consequently, each decoder block consists of only two sub-layers, one masked multi-head attention layer making sure it

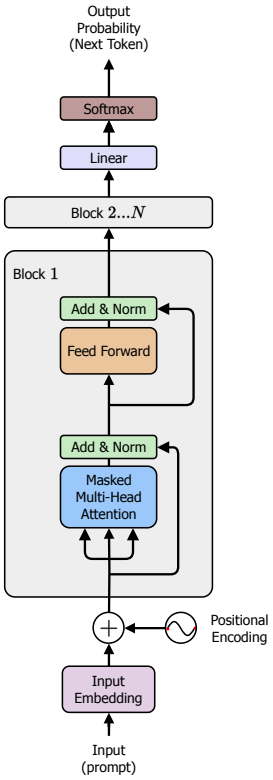


Figure 2.6: Decoder-only Transformer architecture. Blocks are stacked N times to create a deeper network.

only attends on preceding words, and one feed forward neural network layer. When given an input prompt, the model then sends the entire input through the model and generates one token. This token gets appended to the end of the input and sent in again. This process then repeats until a special end token is generated.

2.3.6 (Pre-)Training

The training process of large-scale language models entails significant computational costs (Sharir et al., 2020). This is primarily attributed to the great number of parameters that these models encompass. An essential component of neural network computations is matrix operations (Russell and Norvig, 2020), which are computationally intensive and consequently lead to heavy energy consumption during training and even inference (see Section 2.3.10) (Samsi et al., 2023; McDonald et al., 2022). Moreover, the model's weights and biases undergo iterative updates, a process commonly titled self-supervised learning, to refine the model's performance on the assigned tasks.

Self-supervised Learning

Self-supervised learning is an important technique for pre-training models. This approach capitalises from unlabelled input data, finding patterns and predictability and the learned information can then be used for a supervised learning approach.

Consider a sequence of tokens $s = (t_0, t_1, t_2, t_3, ..., t_i)$, where each token t_j corresponds to a lexical unit, either a word or a sub-word. Each sequence, potentially a sentence from the corpus employed for model training, undergoes a transformation where the last token is removed. The model is then presented with the transformed input sequences, subsequently letting it predict the token distribution of the excised token. By using a loss-function, such as cross-entropy loss, the predicted token distribution is compared to the actual distribution (the predicted token versus the actual token) (Chen et al., 2023b). This provides a foundation in the back-propagating process, updating the weights and biases of the model accordingly. This iterative refinement process persists until all predefined training conditions, for example number of epochs, are satisfied.

2.3.7 Prompt Learning

Prompt learning, also called in-context Learning, is a technique used to adapt a language models' ability to perform better on specific tasks (Liu et al., 2021; Floridi and Chiriatti, 2020). This technique utilises the pre-existing knowledge of large language models by using prompts – contextual instructions – to steer the model's predictions without any extensive retraining of model parameters (Floridi and Chiriatti, 2020). Given that no parameters are updated during prompt learning, this approach typically faces limitations in memory retention and may not achieve optimal task performance, in the given tasks.

Furthermore, in order to steer the prediction of the model in a certain direction, n-shot prompting is frequently used (Chen et al., 2023a). The approach of n-shot prompting involves injecting n number of examples, with a `'Question'` and an `'Answer'`, into the prompt, giving the model temporary knowledge of the problem (Chen et al., 2023a).

2.3.8 Fine-tuning

In order to adapt pre-trained models to new tasks or datasets, *fine-tuning* is used. It involves taking a model that has been trained on a large dataset for a specific task, and then continuing the training process on a smaller dataset or a related task (Ruder, 2016). During fine-tuning, the parameters of the pre-trained model are updated using the new data, allowing the model to learn task-specific features and patterns while retaining the knowledge gained from the original training (Ruder, 2016). Fine-tuning is particularly useful when labelled data for the target task is limited, as it leverages the knowledge encoded in the pre-trained model to improve performance on the new task, which is why self-supervised learning can be utilised (Ruder, 2016). The technique of fine-tuning is commonly used for large and small language models, where the base model typically is very general and pre-trained on vast datasets, and the need to specialise the model in a specific domain persists.

Low-Rank Adaptation (LoRA)

Hu et al. (2021) introduced a novel approach to fine-tune pre-trained deep neural networks, such as Transformers, for tasks constrained by limited labelled data. This methodology, low-rank adaptation, confronts the challenge of fine-tuning large-scale neural networks, or language models, on smaller datasets, frequently resulting in overfitting and sub-optimal generalisation.

Training the entire weight matrix of a model is infeasible. Hu et al. (2021) gained the insight that it is not necessary to update each weight separately and therefore proposes LoRA – a low-rank matrix decomposition of the weight matrix. It is proposed in the forward pass h as

$$h = W_0 x + \Delta W x = W_0 x + W_A W_B x \tag{2.9}$$

where $W_0 \in \mathbb{R}^{d \times k}$ is the pre-trained weight matrix, ΔW is the updated weight matrix which is decomposed into low-rank matrices $W_A \in \mathbb{R}^{d \times r}$ and $W_B \in \mathbb{R}^{r \times k}$ where $r \ll \min(d, k)$. The input in the forward pass h is x (Hu et al., 2021). The regular fine-tuning process and the fine-tuning process using LoRA are depicted in Figures 2.7 and 2.8, respectively. Effectively, this means that a weight update matrix ΔW will be computed from the low-rank matrices W_A and W_B, retain its dimensions $d \times k$, but use up to 10,000 times less amount of parameters than the pre-trained base model (Hu et al., 2021).

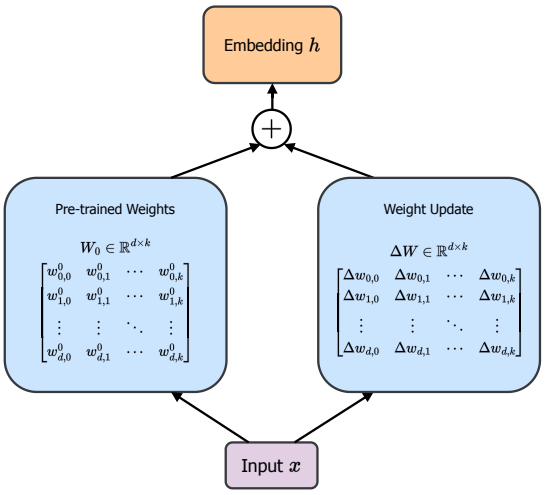


Figure 2.7: The fine-tuning process where a regular weight update $h = W_0 x + \Delta W x$ is computed.

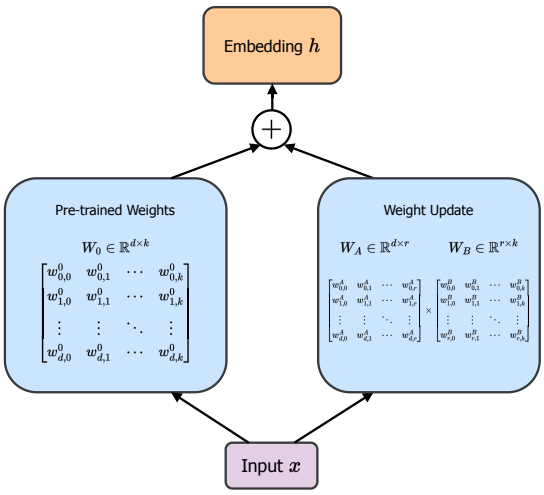


Figure 2.8: The LoRA process where the weight update $h = W_0 x + W_A W_B x$ is computed.

2.3.9 Memory Optimisation Techniques

Reducing memory consumption is always a priority when training neural networks. We introduce two techniques that optimise memory usage when fine-tuning.

Quantised Low-Rank Adaptation (QLoRA)

Further improvement was made to LoRA when Dettmers et al. (2023) introduced quantisation in the context of fine-tuning. This makes the process of fine-tuning with LoRA even more efficient, making it even less computationally demanding to fine-tune language models. Quantisation involves converting data from a high-information representation to a lower-information representation, thus requiring less memory. This typically means reducing the number of bits used, such as converting from 16-bit floats to 4-bit integers. To fully utilise the range of the lower-bit data type, the input data is often rescaled to fit within the target data type range by normalising it based on the absolute maximum value of the input elements, typically organised as a tensor. With this approach QLoRA is defined as

$$h^{\text{BF16}} = \text{dequant}(\text{dequant}(c_1^{\text{FP32}}, c_2^{\text{k-bit}}), W_0^{\text{4bit}})x^{\text{BF16}} + W_A^{\text{BF16}} W_B^{\text{BF16}} x^{\text{BF16}} \tag{2.10}$$

where `dequant()` is the dequantisation process which retrieves the quantised information and c is the quantisation constant, BF16 is a so called brain float with 16 bits and FP32 is a 32-bit floating point tensor.

DeepSpeed

DeepSpeed is an optimisation library developed by Microsoft Research. DeepSpeed offers a suite of features and optimisations aimed at enhancing the efficiency and scalability of training large-scale models. Rajbhandari et al. (2020) introduced the Zero Redundancy Optimiser (ZeRO) to optimise memory usage and accelerate training as a part of DeepSpeed.

ZeRO tackles the memory limitations inherent in training large-scale models by eliminating redundant model states across distributed devices. While this can be achieved with basic data parallelism, ZeRO can partition model states and gradients during training, dismissing redundant memory states during parallelism (Rajbhandari et al., 2020). Complementing ZeRO, DeepSpeed offers a suite of features and optimisations designed to further enhance training efficiency and scalability. One example of this is model parallelism for distributing parameters across multiple devices for improved computational efficiency.

2.3.10 Inference

In the prediction phase often referred to as inference, a trained language model is employed to generate subsequent tokens predicted on a given prompt (Pope et al., 2022). Since the model's inherent architecture is overall sequential, depicted in Figure 2.6, it only has the capability of predicting a singular token. Post-generation, this token is appended to the original prompt, serving as a new basis for subsequent token generation. This iterative process, making it autoregressive, is run until the model produces a special terminal token or a pre-defined threshold is satisfied, e.g. the maximum number of tokens is generated (Pope et al., 2022; Vaswani et al., 2017).

2.3.11 Small Language Models – Phi Models

Small language models (SLMs), such as Phi models, introduced by Gunasekar et al. (2023), are language models with a small parameter count trained on high quality, textbook-like data.

The modest parameter count allows for smart fine-tuning and customisation where resource constraints are a concern (Zhou et al., 2023). The model is what is sometimes referred to as a raw model, meaning that it has only been pre-trained, making its output sometimes unrelated and hallucinatory (Frieske and Shi, 2024). Moreover, as the field of AI progresses the development of SLMs is mainly driven by the need for environmentally sustainable and cost-effective solutions, in contrast to large language models that require enormous amounts of energy to function (Zhou et al., 2023).

Phi models – a series of SLMs – are relatively small compared to other language models. Phi-1.5 achieved a score of 44.6 GSM-8K, 41.4 on HumanEval and 43.5 on MBPP, which are commonly utilised benchmark datasets for evaluation of language models. GSM-8K, or Grade School Math 8K, is a dataset consisting of 8500 diverse grade school math problems (Cobbe et al., 2021), HumanEval is a dataset designed to assess the problem-solving capabilities, specifically code synthesis from strings (Chen et al., 2021) and MBPP, or Mostly Basic Python Problem is a dataset consisting of 1000 Python programming problems evaluating basic fundamental programming concepts (Austin et al., 2021). For reference, GPT3.5 has achieved scores of 78.1, 62.2 and 77.8 respectively. Phi models perform better than much larger language models, such as Llama2-7B and Llama-65B (Li et al., 2023b), which is a testament to its efficiency and the quality of its training data. Phi-1.5 consists of *only* 1.3 billion parameters while its successor Phi-2 consists of 2.7 billion parameters (Microsoft, 2023). On April 22nd 2024, Microsoft published the technical paper announcing Phi-3, the latest iteration of models in the Phi series. Phi-3 has the same underlying architecture as its predecessor, Phi-2, but its training data is altered. The newest model comes in a number of different versions, with the smallest one, Phi-3 mini, consisting of 3.8 billion parameters (Abdin et al., 2024). It achieved a score of 82.5 on GSM-8K, 59.1 on HumanEval and 70.0 on MBPP.

Chapter 3

Datasets

This chapter explains the datasets used in this project, detailing how they were collected, processed, formatted and utilised for both inference and training during the fine-tuning of the Phi-2 model. Section 3.1 goes through the different considerations taken into account, while Sections 3.2 and 3.3 cover the two datasets used.

The Phi-2 model is fine-tuned using two distinctly different datasets. When fine-tuning a pre-trained model such as a Phi model it is important to ensure that the data is specialised and relevant to the task at hand. The goal of fine-tuning a pre-trained model is to increase its expertise in one domain. As the Phi-2 model is already pre-trained with high quality, textbook-like data, it is important that the data used for fine-tuning also adheres to similar levels of quality. The datasets used for this work are introduced in this chapter.

3.1 Considerations

Finding high quality, relevant data for text generation is an important task in the fine-tuning process of a language model, whether small or large. An inherent aspect of fine-tuning is that the data is somewhat niche or specialised, and can not be of any nature. Considering that Phi-2 is pre-trained on high quality, textbook-like data it is also important not to dilute the model with low quality data for the fine-tuning process.

Furthermore, the size of the dataset used for fine-tuning does not necessarily need to be very large. While this may seem far too small for a language model, one must consider the task at hand. To fine-tune a pre-trained language model not as much training data is needed before the base model starts to converge to a new setting. As the base model is pre-trained, it is already trained on large amounts of data. On a side note, smaller datasets also help with the easing of memory requirements during the training process. OpenAI, the leading LLM developer, provides a fine-tuning guide of their models, the GPT class. In OpenAI's guide,

it is suggested that when fine-tuning GPT3.5 and GPT4, one should use at least 10 examples. Also, improvements from fine-tuning are typically seen when the training dataset has 50 to 100 entries. In addition to this, OpenAI state that the correct number can vary greatly (OpenAI, 2024).

3.2 OptiGuide Benchmark Datasets

A number of benchmark datasets are provided by Li et al. (2023a) for OptiGuide. These smaller datasets are suitable for evaluating if SLMs like the Phi-2 model can be fine-tuned for what-if analysis within the field of supply chain optimisation. In this thesis, we are using the benchmarks' questions and answers for our fine-tuning. There are six datasets in total based on different settings and scenarios. One of the datasets, `coffee.benchmark.json`, is a benchmark dataset that is provided by Microsoft Research, and one for which Li et al. (2023a) have implemented the supply chain optimisation model. The other examples, including a TSP model with benchmark questions and answers, are provided along with code taken from Gurobi source code examples (Gurobi, 2024). This means that this work can easily be extended with other datasets.

The datasets are relatively small, compared to the datasets which the model is pre-trained on, containing a couple of hundred data points. Referring to the considerations done in Section 3.1, this size is not an issue, but rather fitting for the fine-tuning task. Being a benchmark dataset, it is especially suitable for comparison among language models and provides a result for further research on Phi-2 or other SLMs and LLMs. Furthermore, combining all models' benchmarks results in a dataset of substantial size.

The model benchmark datasets are for the purpose of our research expanded with the model source code, implemented in Python. The dataset is structured in a Question-Answer (QA) format in a JSON file, where the Question corresponds to a prompt, and the Answer corresponds to a completion. A key is defined for both the prompt and the completion, and these are naturally `'Question'` and `'Answer'`, which is the structure of each row of the dataset.

3.2.1 Data Samples and Statistics

The source code for the models used in the OptiGuide benchmark datasets are retrieved directly from Gurobi Python examples (Gurobi, 2024). The relevant model source code is concatenated to the question to add context for the SLM. In Table 3.1 we look at some samples of questions and answers from the dataset. In total, the dataset has questions and answers for six different models, corresponding to the number of benchmark datasets, and these are

- `diet.py`
- `facility.py`
- `netflow.py`
- `tsp.py`

- `workforce1.py`
- `coffee.py`

where `coffee.py` is developed by Microsoft Research for OptiGuide (Li et al., 2023a).

Context	Question	Answer
Given the following code depicting a supply chain or optimisation problem …`categories, minNutrition, maxNutrition = gp.multidict( {"calories": [1800, 2200],` …	Is it possible to steer clear of macaroni?	`m.addConstr( buy["macaroni"] == 0, "_")`
Given the following code depicting a supply chain or optimisation problem …`capacity_in_supplier = {'supplier1': 150, 'supplier2': 50, 'supplier3': 100}` …	What is the impact of `cafe3` exclusively purchasing coffee from `roastery1` and `roastery1` solely selling its coffee to `cafe3`?	`for c in cafes:\n\t if c = "cafe3":\n\t\t m.addConstr( y_light["roastery1", c] == 0, "_")\n\t\t m.addConstr( y_dark["roastery1",` …
Given the following code depicting a supply chain or optimisation problem …`commodities = ["Pencils", "Pens"] \n nodes = ["Detroit", "Denver", "Boston", "New York", "Seattle"] \n arcs, capacity = gp.multidict( {("Detroit", "Boston"): 100,` …	What would happen if the shipping charges to Boston increased by 9 dollars?	`for a in commodities:\n\t for b in nodes:\n\t\t if (a, b, "Boston") in cost:\n cost[a, b, "Boston"] += 9`

Table 3.1: Samples from the OptiGuide benchmark dataset with questions, including context, and answers. The samples are from `diet.py`, `coffee.py`, `netflow.py` in that order.

The full dataset contains 1419 samples, which have been split into 993 for training, 227 for validation and 199 for testing. This corresponds to 70%, 16% and 14% respectively. Question types range from variations in demand, fluctuations in transport costs and changes in demand to numerous other factors that impact the specific supply chain or optimisation problem.

As outlined in Section 2.1, these problems are typically constructed as mixed integer programs in a constraint based approach. By utilising the Gurobi Python models of these six problems, we seek to understand if a fine-tuned SLM can be used for code generation as a part of a holistic framework for what-if analysis. Providing the correct code snippet to update the supply chain model, or optimisation problem, a model may act as a useful tool for supply chain managers. Importantly, questions are asked in natural language, and answered in Python code that would be added to the original supply chain model. Incorporating a proficient model in the workflow of optimising supply chains may help streamline what-if analysis.

3.3 Job Shop Scheduling Dataset

In this section, we explore the dataset used to fine-tune the model for the job shop scheduling problem (JSSP).

Using a large, balanced, high-quality dataset is arguably the most important aspect when trying to teach a model new knowledge. In this problem it is important that the model gets an understanding of the intricate aspects of it and learns how to actually solve them. Therefore, we created our own data, consisting of 5000 instances of the JSSP. When creating the instances we followed a similar format to Taillard (1993). An example of the format used can be seen below:

```
7  7
4  32  0  27  1  35  3  91  2  28  6  68  5  73
3  71  4  92  5  92  6  58  1  92  2  28  0  31
0  26  2  94  4  32  5  35  3  78  1  65  6  73
1  47  4  22  2  36  6  55  5  44  0  60  3  66
3  69  5  91  1  21  4  96  6  90  2  67  0  34
2  73  5  61  0  45  4  51  3  87  6  39  1  90
4  50  0  73  5  49  6  73  2  45  3  38  1  89
```

where the first line defines the number of machines and number of jobs. After that there is one line per job where each line defines the tasks for that specific job. The tasks are grouped by two numbers, what machine, m_i, it has to be run on and t, how long time that task takes. All examples were randomly generated with values shown in Table 3.2.

Furthermore, for each sample a solution was derived using Google's open-source solver, OR-tools. The solution consists of two parts; the first being the optimal makespan – total processing time, for the problem and secondly, a detailed schedule of all tasks. Furthermore, the dataset was partitioned into training, validation, and test sets, with the training set comprising 70%, the validation set 15%, and the test set 15%. Lastly, each sample was formatted into the QA format. Each instance was formulated as a question, encompassing both the problem's constraints and objectives, along with the actual problem description. Similarly to the OptiGuide benchmark datasets, keys are defined as `'Question'` and `'Answer'`. This is done to ensure that the model comprehends the problem thoroughly, taking into account all relevant constraints of the structured data.

3.3.1 Prompt Learning Data

In order to apply prompt learning, the test set was transformed. Each sample was encapsulated with the identical problem formulation, as seen in 3.1, and constraints as those present in the original formatting technique. However, to imbue the prompt with relevant knowledge, we inserted a pair of illustrative examples, complete with problem statements and corresponding solutions. This test set is therefore used when any reference to 2-shot prompting is made, within JSSP. Likewise, 0-shot prompting indicates the usage of the original test set.

```
Solve the following Scheduling Problem where the first line contains
two numbers:  number of jobs, n_jobs, and number of machines,
n_machines.  Then follows n_jobs lines, one for each job.  Each job
consists of n_machines tasks where each task consists of two numbers;
what machine it needs to be run on and how many time units it takes.
The machines are indexed starting from 0.  Each job has a sequence
of tasks that must be run on specific machines for a certain
duration.  Each machine can only run one task at a time, and tasks
for a job must be run in the order given.  The goal is to minimize
the total processing time (makespan):  Problem:
```

Figure 3.1: Problem formulation for JSSP. In a 0-shot setting, only the problem instance is appended to the end after `Problem:` . In a 2-shot setting, two example instances with solutions are appended as well as the instance to be solved.

Type	Range
Number of machines	5 – 12
Number of jobs	5 – 12
Processing times for a task	20 – 100

Table 3.2: Ranges used to generate instances for the job shop scheduling problem (JSSP). Each instance is characterized by three types of variables: the number of machines, the number of jobs, and task processing times. These variables are sampled from the specified range, with an evenly spread distribution

Chapter 4

Method

This chapter provides an end-to-end overview and in-depth description of the steps taken to conduct the research and reach the results presented in this thesis. The chapter begins with a small note on the process of handling data, described in Section 4.1. After this section follows an explanation of the procedure of prompting the base model, described in Section 4.2. Moreover, the process of fine-tuning a language model and the requirements that come with it are described in Section 4.3. Finally, Section 4.4 describes the evaluation of the resulting outcomes.

Fine-tuning a language model is a process that draws heavily from the theoretical concepts discussed in Chapter 2, Theoretical Background. It involves adapting a pre-trained model to perform specific tasks to excel in particular domains. This chapter delves into the specifics of the fine-tuning process, providing a detailed road map for leveraging the Phi-2 model effectively.

In essence, fine-tuning involves understanding the intricacies of the model's architecture and adapting it to address the targeted objectives. The approach outlined in this section encapsulates a five-step process designed to streamline the fine-tuning process, ensuring theoretical foundations and the practical implementation are aligned.

By adhering to this methodological framework, the aim is to not only optimise the performance of the fine-tuned Phi-2 model on specific tasks in a supply chain context, but also to showcase the rationale behind each step taken in the fine-tuning process. With the help of this methodological framework, this chapter aims to provide a comprehensive understanding of the strategies used and their implications for the performance of the fine-tuned Phi-2 model.

4.1 Data Gathering and Formatting

The datasets we use in this thesis are introduced in Section 3. We opt to use open source data in this project. Using confidential or proprietary data can limit transparency, while using open source data allows for benchmarking and reproduceability. In order to adhere to the QA format suitable for text generation and prompting in general, and Phi-2 in particular, we format the data to produce datasets suitable for the task of fine-tuning an SLM. Similar formatting is done for both the OptiGuide benchmark datasets and the JSSP dataset.

4.2 Base Model

In order to evaluate and assess the performance of the fine-tuned Phi-2 models, it is essential to have an understanding of the performance of the base Phi-2 model. Although this is not a comparative study, and the Phi-2 model is primarily designed specifically for fine-tuning, we undertake a preliminary analysis on the base model to establish a reference point. This analysis includes 2-shot prompting to gauge the model's initial capabilities before fine-tuning.

4.2.1 Baseline Prompting

In order to investigate the impact of fine-tuning the model in the two areas handled in this thesis, a baseline result is of interest. This baseline serves as a reference point for assessing the effectiveness of subsequent fine-tuning efforts.

The initial step in the process of prompt learning involves formatting prompts in order to encapsulate context, questions and relevant information in order for the model to generate a relevant answer (see Section 3.3.1 for further details). In the second step the model is inferred with the crafted prompts. Lastly, we evaluate the generated output using metrics defined in Section 4.4.

4.3 Fine-tuned Models

Exploring fine-tuned versions of Phi-2 is the main research interest of this thesis. Therefore, it is essential to provide a detailed explanation of the fine-tuning process and outline the associated requirements thoroughly.

4.3.1 Fine-tuning Process

The process of fine-tuning Phi-2 is a five-step process. These five steps ensure the pre-trained model is leveraged in a relatively inexpensive, efficient and useful way. We present a high-level overview of the steps taken in order to fine-tune the Phi-2 model:

1. **Find relevant data.** Ensuring that the model is fine-tuned on relevant, high-quality data is vital for performance. Reliable training data for the task at hand must be used to provide the necessary context and information for the model to learn effectively. In

this thesis, benchmark datasets for what-if scenarios in supply chain optimisation are used in one scenario and JSSP instances in another.

2. **Ensure infrastructure requirements are met.** In order to initiate the fine-tuning process, it is essential to ensure that the infrastructure meets the necessary requirements for efficient training. This includes having access to suitable hardware resources such as a graphics processing unit (GPU) and sufficient storage capacity for the dataset and the base model.

3. **Use techniques to limit memory usage.** Fine-tuning a language model like Phi-2 can be resource-intensive, particularly in terms of memory usage. Even though Phi-2 is categorised as a small language model, it still has 2.7 billion parameters. Employing techniques such as quantisation and using a smaller dataset helps manage memory usage effectively.

4. **Train the relevant layers.** During fine-tuning, it is crucial to selectively update the parameters of the pre-trained model to adapt it to the specific task or domain. Typically, only a subset of layers in the model are fine-tuned, while the rest are kept frozen to retain the knowledge learned during pre-training. We select the layers to be trained on the fine-tuning dataset.

5. **Evaluate the results.** Once the fine-tuning process is complete, it is important to evaluate the performance of the tuned model on a validation set or through other appropriate metrics. Furthermore, we look at different metrics such as ROUGE and BLEU to evaluate the performance. This helps assess whether the fine-tuning process has led to improvements in model performance in the specific domain and whether any further adjustments are necessary. Also, the fine-tuned model can be loaded and used for prompting.

4.3.2 Infrastructure

For any project, it is essential to carefully consider the infrastructure for building, training and deploying. This becomes even more crucial when dealing with machine learning projects that involve models containing billions of parameters.

Azure

The entire project was hosted on Microsoft Azure, utilising its robust cloud computing infrastructure. The implementation, comprising of training and evaluation, was done using Python. Python, often used together with Jupyter Notebook, is a tool with which most modern language models are designed, trained and evaluated. This way of working is prevalent in the field of machine learning, especially for natural language processing tasks. Microsoft Azure provide a comprehensive set of services suitable for this project and workflow, along with offering impressive scalability and reliability.

Firstly, modern neural networks, including models based on the Transformer architecture, require a great amount of computing power. Language models, small or large, typically require GPU (see Section 4.3.2) for training. Azure streamlines access to GPUs hosted as a cloud

service by Microsoft through a virtual machine (VM). This enables us as students to conduct research which involves computationally heavy tasks, such as training and fine-tuning language models. Access to computing resources can easily be scaled up or down which adds flexibility to the project. For the purpose of the tasks that are performed in this thesis, it offers a similar, but far more holistic, service similar to the popular Google Colaboratory in Google Drive.

Azure provides many services such as Kubernetes or Docker which adds flexibility and ensures projects can be hosted end to end on Azure. In a sense, Azure is an end-to-end platform for hosting machine learning projects. Azure also offers source control and the use of pipelines. Furthermore, one has direct access to many language models in Azure via the Azure Machine Learning portal.

Furthermore, Azure Machine Learning Studio offers several built-in tools for fine-tuning and evaluation, where hyperparameters are easily set and changed. The training process is set up as a pipeline within Azure to make use of the chosen compute instance, which in our case is a specific GPU. The trained model can then be registered and deployed as endpoint within Azure to be used for inference. The full model weights can also be directly uploaded to Hugging Face or downloaded. For our research purposes, this is not something we utilised. The base model is simply loaded from Hugging Face, a machine learning platform and community where one can find all the latest models and architectures from the leading AI companies. The Hugging Face identifier for Phi-2 is `microsoft/phi-2`.

Hosting the project on Azure provides a scalable, reliable, and efficient infrastructure, enabling training, data handling and model deployment throughout the lifespan of this project. Most importantly, it allows us to leverage the vast computational resources needed for training.

Graphics Processing Unit (GPU)

Graphics processing units are specialised hardware components designed to handle and accelerate the rendering of images and graphics in computers. Originally developed for gaming and graphical applications, GPUs have found extensive use in artificial intelligence.

GPUs play a crucial role in training and running complex models, such as large language models. One of the main research interests of fine-tuning Phi-2 is the relatively small complexity of the model compared to the pre-training of LLMs like GPT4 that requires large amounts of GPU computing resources. These requirements make dealing with LLMs very costly and thus very cumbersome for anyone – individual or organisation – other than the biggest technology companies. Unlike traditional central processing units (CPUs), GPUs are highly parallelised, meaning they can perform multiple computations simultaneously, making them ideal for handling the massive amounts of matrix multiplications and neural network computations involved in training Phi-2.

In the fine-tuning process of Phi-2, a number of layers are selected to be retrained. Phi-2 is a model with 2.7 billion parameters, meaning that hundreds of millions of parameters are fine-tuned during training. This requires performing numerous matrix operations and back-propagation steps, which is computationally very intensive.

For the training conducted during the fine-tuning of Phi-2 we are using a scalable compute cluster from NVIDIA, with its specifications outlined in Table 4.1. The training time using this GPU for the fine-tuning process is presented in Table 4.2 and Table 4.3.

vCPU	Memory	Temp Disk	NVMe Disks	GPU	GPU Memory
24	220 GiB	64 GiB	960 GB	1	80 GiB

Table 4.1: Specifications for compute cluster `Standard_NC24ads_A100_v4` used in Azure for the fine-tuning of Phi-2 models. This compute cluster has 24 vCPU cores with 220 GiB RAM and one GPU with 80 GiB RAM. The GPU is an NVIDIA A100 PCIe Tensor Core GPU. The computer cluster also has 64 GiB temporary disk storage and 960 GB NVMe (Non-Volatile Memory express) SSD (Solid-State Drive) storage.

4.3.3 Memory Optimisation Techniques in Python

We describe how the memory optimisation techniques introduced in Section 2.3.9 and LoRA, introduced in Section 2.3.8, are used in Python.

Quantisation

To effectively reduce the memory usage during the fine-tuning of the model, the model is quantised during training. The model can be quantised to a set number of bits as shown by the results of QLoRA (see Section 2.3.9). Quantisation in Python was done using the `bitsandbytes` library (HuggingFace, 2024). In consideration with the GPU architecture used, depending on during which stage of the project the training was conducted, floating point tensors were used, rather than brain floats.

DeepSpeed

The techniques of DeepSpeed, outlined in Section 2.3.9, are applied during the training phase. In Azure, there is a choice between stage 2 and stage 3 DeepSpeed, where stage 2 enables parallelism optimisation and stage 3 further memory reduction through techniques such as parameter partitioning. During training in this project, stage 2 was used in Azure.

LoRA in Python

To fine-tune the model, we select the layers that are to be trained with the fine-tuning data. Generally, one only re-trains a number of the last layers to fine-tune a base model. We train

the `q`, `v`, `fc1` and `fc2` layers during fine-tuning. We do this using LoRA, a technique outlined in Section 2.3.8. When fine-tuning using LoRA, it is central that a rank, r, is determined before the training commences. Furthermore, LoRA usage is enabled through the `PEFT` (Parameter-Efficient Fine-Tuning) library in Python. Also, LoRA alpha, α, is defined as a spreading factor. The LoRA alpha parameter is typically set to 1 or 2 times the value of the LoRA rank, r.

4.3.4 Training Process

During fine-tuning, the layers specified in Section 4.3.3 are trained in a similar form to how they were (pre-)trained for the base model. In light of this, the same considerations as when training a base model must be taken. Parameters such as number of epochs, learning rate and batch size need to be determined. Furthermore, fine-tuning and memory optimisation related parameters such as LoRA r and DeepSpeed need to be determined. Also, the optimiser is chosen as the AdamW optimiser, a derivative of the popular Adam (Adaptive Moment Estimation) optimiser that incorporates weight decay into the optimisation process (Loshchilov and Hutter, 2017).

It is important to note that training has been run several times, with different parameter setups, to achieve the best results. Due to the heavy computational burden and requirements, we are not able to perform a grid search or use other techniques to find the optimal parameter and hyperparameter setup. The values of the parameters presented in Tables 4.2 and 4.3 are the parameters used for the runs that achieved the best results. For instance, these runs do not utilise QLoRA. Instead, the models are trained using LoRA and DeepSpeed techniques. In the same manner, the decaying rate of the weights is zero, essentially meaning an Adam optimiser was used as opposed to an AdamW optimiser.

We used the parameters and hyperparameters presented in Table 4.2 for the fine-tuning task on the OptiGuide benchmark datasets while Table 4.3 presents the parameter and hyperparameter values used for the fine-tuning task on the JSSP dataset.

Parameter	Value
Epochs	5
LoRA alpha α	16
LoRA rank r	16
LoRA dropout	0.15
Batch size	10
Max sequence length	2048
Learning rate	0.0003
DeepSpeed stage	2
Linear warmup steps	20
Weight decay	0
AdamW β_1	0.9
AdamW β_2	0.999
AdamW ϵ	1e-8
Evaluation steps	500
Training time	**2h**

Table 4.2: Parameter and hyperparameter setup for the fine-tuning task of Phi-2 on OptiGuide benchmark datasets. The training time is also included.

4.4 Evaluation

The fine-tuning task involves specialising the model in a particular subject or task, requiring an evaluation accommodating for this fact. Commonly seen in evaluation of the general performance of LLMs and SLMs, a plethora of tasks and benchmarks are used. For our purposes, we use a couple of relevant and widely recognised metrics to assess the performance of both the base Phi-2 model and the two fine-tuned Phi-2 models. This approach ensures a nuanced evaluation of the results of this thesis.

4.4.1 Metrics

In order to evaluate both the base Phi-2 model and the two fine-tuned Phi-2 models, we employ a number of metrics. For the OptiGuide benchmark datasets, we calculate the average BLEU and ROUGE scores in a 0-shot setting on the test set for both the base model and the fine-tuned model. We apply the same evaluation metrics on the base model and the model fine-tuned for JSSP. The base model is prompted in a 2-shot setting and the fine-tuned model is prompted in both a 0-shot and a 2-shot setting on the test set.

BLEU

BLEU (Bilingual Evaluation Understudy) score is a metric introduced by Papineni et al. (2002). It is often employed to evaluate the quality of machine-generated translations, primarily in natural language processing tasks like machine translation. However, it can also find application in code generation tasks. In this context, the generated code is compared against one or more reference codes to measure their similarity. BLEU computes precision

Parameter	Value
Epochs	5
LoRA alpha α	32
LoRA rank r	32
LoRA dropout	0.15
Batch size	64
Max sequence length	2048
Learning rate	0.0003
DeepSpeed stage	2
Linear warmup steps	20
Weight decay	0
AdamW β_1	0.9
AdamW β_2	0.999
AdamW ϵ	1e-8
Evaluation steps	500
Training time	**8h**

Table 4.3: Parameter and hyperparameter setup for the fine-tuning task of Phi-2 on JSSP. The training time is also included.

by tallying the number of overlapping n-grams (sequential sequences of n tokens) between the generated and reference codes. We calculate an average BLEU score based on the scores for BLEU-1, BLEU-2, BLEU-3 and BLEU-4, corresponding to the BLEU score for unigrams, bigrams, trigrams, and quadgrams.

Additionally, it incorporates a brevity penalty to address the tendency of machine-generated code to be shorter than reference code (Papineni et al., 2002). Similar to its use in translation tasks, a higher BLEU score in code generation suggests a closer match between the generated and reference codes. Despite its utility, BLEU has limitations, including its reliance on n-gram precision and insensitivity to code semantics, necessitating the employment of supplementary evaluation methods for a comprehensive evaluation of code generation quality (Papineni et al., 2002).

ROUGE

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics introduced by Lin (2004). The metrics are widely employed to assess the quality of machine-generated text, with applications spanning tasks such as text summarisation and machine translation. It evaluates the degree of overlap between the generated text and reference summaries, utilising measures like n-gram overlap and word overlap. ROUGE computes precision, recall, and F1-score, offering valuable insights into the effectiveness of the generated text (Lin, 2004). We calculate the average ROUGE score based on the scores for ROUGE-1 and ROUGE-2, corresponding to the ROUGE score for unigrams and bigrams.

In addition to the traditional ROUGE metrics, ROUGE-L specifically focuses on the longest common subsequence (LCS) between the model-generated text and the reference. This met-

ric places a greater emphasis on capturing the order of elements in the generated text compared to other ROUGE variants, providing a more nuanced evaluation of the output's coherence (Lin, 2004). While traditional ROUGE metrics offer a comprehensive evaluation of text quality, ROUGE-L offers additional insights into the model's ability to produce output that closely resembles the reference in terms of order and structure.

4.4.2 Evaluating the OptiGuide Benchmark Datasets

For the fine-tuning task on the OptiGuide benchmark datasets, we evaluate the performance using the metrics outlined in the above sections. The BLEU score is reported both for the 0-shot prompts given to the base Phi-2 model, as well as for the entire test set for the fine-tuned Phi-2 model. Additionally, we report the ROUGE score along with the the ROUGE-L score. We use the Phi-2 pre-trained tokenizer for the base model performance evaluation and an updated tokenizer for the fine-tuned Phi-2 model. Finally, we conduct systematic tests on the generated Python code to verify that it is executable.

4.4.3 Evaluating the Job Shop Scheduling Dataset

We evaluate the performance on the job shop scheduling problem by calculating the above mentioned metrics. In order to acquire a baseline score, the base model is evaluated with a 2-shot setting on the test set. The fine-tuned model is then evaluated on the test set, both with a 0-shot setting and a 2-shot setting. Moreover, a core detail in BLEU score is its dependence on n-grams. Therefore, when calculating the score, it is of great importance how the strings are tokenized. We use two different tokenizers; a white-space tokenizer and the Phi-2 pre-trained tokenizer which ultimately is a byte-pair encoding (BPE) tokenizer. A correct solution is very much dependent on the time intervals and specific task numbers, meaning that a white-space tokenizer retains the ability to accurately tokenize the strings. However, it is still of interest to also observe the BLEU score using the Phi-2 tokenizer. As a final evaluation metric we report the average makespan difference.

Chapter 5
Results

This chapter provides the results for the research conducted in this thesis. The results are structured as two distinct sections, one for each task. Section 5.1 showcases the results for the OptiGuide benchmark datasets and Section 5.2 showcases the results for the JSSP dataset. This chapter provides results for the experiments done following the methodology in Chapter 4. The results lay the foundation for the discussion and conclusion, and are commented briefly in this chapter.

5.1 OptiGuide Benchmark Datasets

As outlined in Section 4.4 we evaluate the performance of Phi-2 on the OptiGuide benchmark datasets in two settings. Firstly, we evaluate the base Phi-2 model performance on 0-shot prompts for the entire test set. Then, we fine-tune the model on the training dataset defined in Section 3 and lastly we evaluate, again, the performance of the fine-tuned Phi-2 model on the test set. We use the metrics defined in Section 4.4. The first metric is BLEU, which is an average score of BLEU-1, BLEU-2, BLEU-3 and BLEU-4. Secondly, we use ROUGE, which is an average of ROUGE-1 and ROUGE-2. Thirdly, we use ROUGE-L and lastly we report the accuracy as defined in Section 4.4.

Table 5.1 and Figure 5.1 present the summarised results. The BLEU score for the 0-shot prompted base model is very low at only 1.34%. The fine-tuned model on the other hand performs much better with an impressive BLEU score of 98.6%. A very high BLEU score on a code generation task means that nearly all the n-grams in the generated code match the reference answer code. Simultaneously, the ROUGE score for the 0-shot prompted base model is also low at 7.14%, while the fine-tuned model is scored at 99.4%. Furthermore, the ROUGE-L score lands at 9.55% and 99.4% for the base model and the fine-tuned model respectively. In summary, the results differ vastly between the base model indicating that fine-tuning the model clearly leverages the underlying strengths of Phi-2. Importantly, we manually confirm

that the generated Python code is syntactically correct and executable. This is the case for the entire test set.

Phi-2 model	BLEU	ROUGE	ROUGE-L
Baseline 0-shot	1.34	7.14	9.55
Fine-tuned 0-shot	98.6	99.4	99.4

Table 5.1: The observed results on the OptiGuide benchmark datasets. The baseline score was calculated on the entire test set using 0-shot prompts.

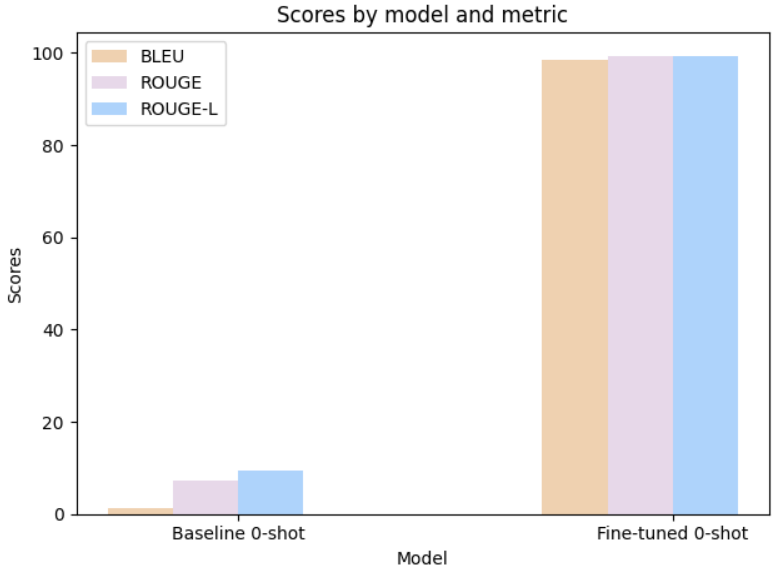


Figure 5.1: Visualisation of the scores presented in Table 5.1.

We also observe that the training loss decreases and converges to a very low value of around 1% after five epochs. Figure 5.2 depicts the evolution of the training loss during the five epochs of fine-tuning of the Phi-2 model. We observe the same result for the validation loss, which also consistently decreases. After five epochs, the validation loss is 1.45% as seen in Figure 5.3. By systematically comparing the ground truths with the predictions, we conclude that 100% of the generated code can be added to the supply chain model code. This means that all code generated is executable.

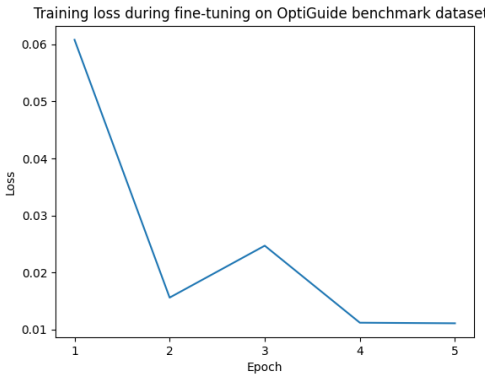


Figure 5.2: Training loss during fine-tuning of the Phi-2 model on the OptiGuide benchmark datasets.

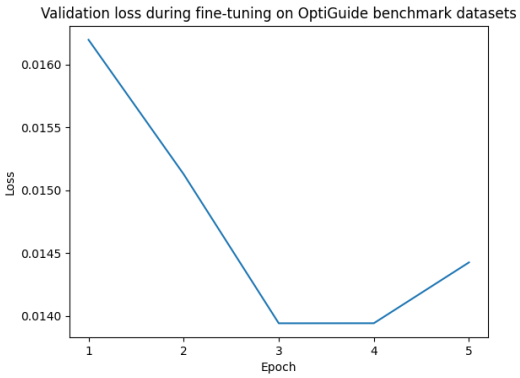


Figure 5.3: Validation loss during fine-tuning of the Phi-2 model on the OptiGuide benchmark datasets.

5.2 Job Shop Scheduling Dataset

We evaluate the performance as stated in Section 4.4 and report the obtained results for the JSSP dataset in Table 5.2 and Figure 5.4. The baseline Phi-2 model, with a 2-shot setting, shows modest results across all scores: BLEU with white-space tokenizer at 0.53, BLEU with Phi tokenizer at 41.0, ROUGE at 46.6, ROUGE-L at 39.3 and finally an average makespan difference of 101. We observe a significant improvement for the fine-tuned model with a 0-shot setting. The scores improved dramatically; BLEU with white-space tokenizer at 21.7, BLEU with Phi tokenizer at 72.4, ROUGE at 68.9, ROUGE-L at 63.1 and finally an average makespan difference of 40. However, the fine-tuned model in a 2-shot setting shows scores slightly better than the baseline; BLEU with white-space tokenizer at 5.00, BLEU with Phi tokenizer at 42.8, ROUGE at 48.6, ROUGE-L at 42.3 and finally an average makespan difference of 67. As stated in Section 5.1, both the BLEU and ROUGE score are averages. However, even though the performance greatly increased for the fine-tuned model, it is important to note that it was not able to fully complete any of the problem instances. The model consistently manages to schedule only a couple of the first tasks and machines correctly before making an error.

Furthermore, Figure 5.5 and Figure 5.6 depict training and validation loss respectively during the fine-tuning process of Phi-2 on JSSP. We can observe that the training loss drastically reduces during training. The validation loss follows the steep decrease of the training loss during the entire training.

Phi-2 model	BLEU WS	BLEU Phi	ROUGE	ROUGE-L	Makespan Diff. (Avg.)
Baseline 2-shot	0.53	41.0	46.6	39.3	101
Fine-tuned 0-shot	21.7	72.4	68.9	63.1	40
Fine-tuned 2-shot	5.00	42.8	48.6	42.3	67

Table 5.2: The observed results on the JSSP test-dataset. WS refers to white-space tokenizer, and Phi to the Phi tokenizer. The scores for BLEU, ROUGE and ROUGE-L are percentages and average makespan difference is the difference between the optimal answer and the predicted answer.

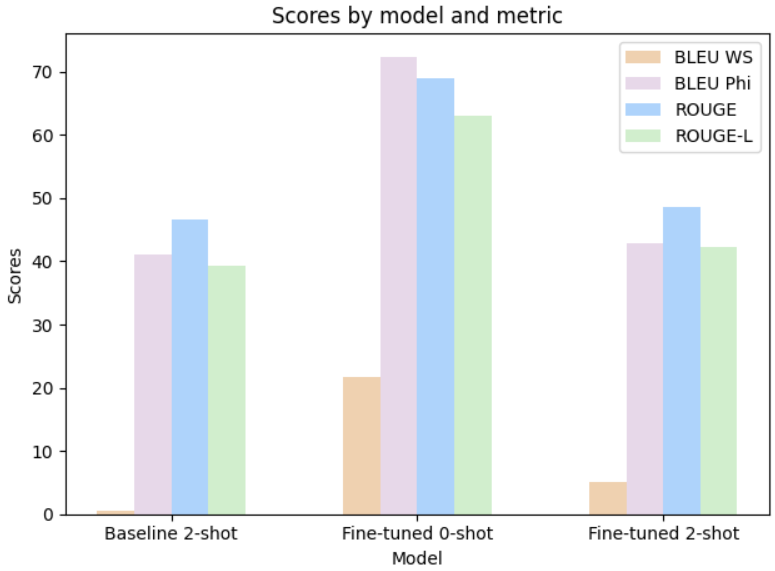


Figure 5.4: Visualisation of the scores presented in Table 5.2. Note that average makespan difference is not visualised here because it is not measured in percentages.

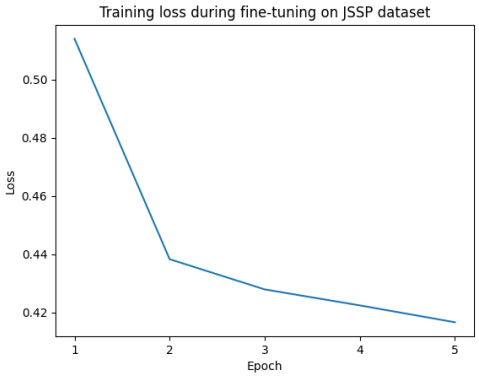


Figure 5.5: Training loss during fine-tuning of the Phi-2 model on the JSSP dataset.

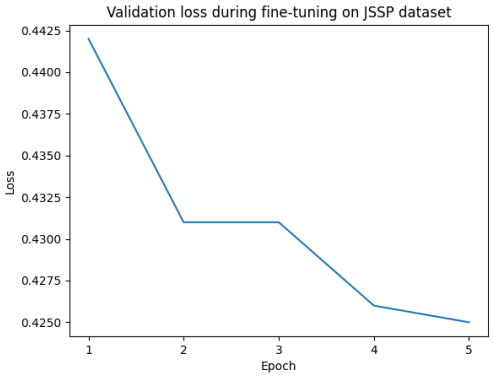


Figure 5.6: Validation loss during fine-tuning of the Phi-2 model on the JSSP dataset.

Chapter 6
Discussion

This chapter further elaborates on the results presented in Chapter 5 and provides thoughts on the implications, considerations and significance regarding the results. Section 6.1 provides an analysis of these factors along with underlying reasons for the outcome of the results, for both datasets. Section 6.2 elaborates further on the performance of SLMs and the prospect of utilising them for fine-tuning. Finally, Section 6.3 covers the considerations and limitations of our work. The discussion and its outcomes provide the necessary background for the conclusions drawn in Chapter 7.

6.1 Implications of Results

The results, presented in Chapter 5, provide insight into the performance of both the base Phi-2 model and fine-tuned Phi-2 iterations on two different tasks in supply chain optimisation. In a time of ever-growing models boosting parameter sizes in the trillions, SLMs such as Phi models can help accelerate the democratisation of AI in general and language models in particular. As we hypothesised in the first phase of this project, smaller models such as the Phi models benefit from fine-tuning and can become diligent performers of specific tasks once fine-tuned. The results we achieve strongly support this.

The fine-tuned models heavily outperform the base Phi-2 model, indicating the fine-tuning to be successful. While the metrics we use to evaluate the performance of the models are relevant and frequently used in both research and industry, we dive deeper into the analysis by interpreting the results. A high BLEU score or ROUGE score indicates that the n-grams generated by the model resemble the reference answer. For many tasks, including code writing, there are several different viable solutions to a given task. However, these metrics do not capture this aspect to an extent, which can possibly imply a poor BLEU score. We present results with considerably high scores for the fine-tuned iterations of the Phi-2 model, meaning our results seemingly do not suffer from this phenomenon.

Additionally, we assess how well the model performs on two very different tasks, which can lead to significant variations in results. For instance, while the base Phi-2 model undergoes some pre-training for Python code generation, it lacks training specifically for solving optimisation problems. While this is an indicator of good performance on code generation tasks, it is essential to consider other factors when assessing performance using predefined metrics. One such factor, among many, that can influence the model's performance is the tendency for the model to hallucinate. This is a phenomenon observed not only in larger models but particularly prevalent in the Phi-2 model.

When assessing model performance with metrics such as BLEU and ROUGE, performance drops rapidly when the output from the model is not formatted as expected. This is one of the factors behind the large jump in performance from base model to fine-tuned. Another factor is that the base model tends to repeat its answers, seemingly indefinitely or until its token size window is filled. These factors are not a discovery of this work, but rather known facts provided by the authors of the model. Microsoft informs that the base Phi-2 model suffers from unreliable responses to instruction, from which it follows that the model can struggle or fail to adhere to nuanced instructions. The base model has a limited scope for code, as it has only been pre-trained on Python code and common packages within Python. Finally, the model suffers from verbosity and hallucination. Microsoft state that Phi-2 often produces irrelevant or extra text following its first answer. The authors believe this is due to the textbook based training data (Microsoft, 2024).

6.1.1 OptiGuide Benchmark Datasets

The results observed on the code generation task for the OptiGuide benchmark datasets differ greatly between the base model and the fine-tuned model in terms of the predefined metrics. As seen in Section 5, the base model has a BLEU score of a mere 1.34%, which is evidently very low. For this task, the factors presented above, in Section 6.1, strongly affect the performance of the base model. On the other hand, as hypothesised, fine-tuning the Phi-2 model for this specific task improves the performance vastly. The fine-tuned model answers nearly every query correctly, considering both the metrics BLEU and ROUGE as well as the executability of the generated code. As reported, the BLEU score for the fine-tuned model is 98.6%. This is a huge leap in performance from the base Phi-2 model.

The results indicate that fine-tuning is very efficient and improves performance greatly. When the model is fine-tuned on the OptiGuide benchmark datasets, this is clearly true. The fine-tuned model performs very well on unseen questions for the six different models in the benchmark. During fine-tuning, the model has gained the ability to limit verbosity and the tendency to hallucinate. It very diligently answers with only the new Python code needed, and no further explanations. This result is important when considering that such a fine-tuned SLM could serve as one part of a larger end-to-end framework or system, where the fine-tuned model's output is to be passed on to another part of the system. In this kind of scenario, it is important to filter out the noise that the base model's verbosity and hallucinations generates.

Given that the Phi-2 model undergoes pre-training specifically on Python code, it is hardly surprising that the fine-tuned model swiftly attains proficiency in generating new Python code in a supply chain setting for what-if analysis. Our findings suggest that through fine-tuning, the model adapts to a contextualised set of tasks, demonstrating its ability to rapidly converge to correct answers. This outcome provides valuable insight on the potential of fine-tuning SLMs. With this insight, organisations and individuals seeking to leverage generative AI capabilities for code generation may find interest in fine-tuning Phi models for their tasks.

For this task, we must also consider the possibility that the model overfits to the training data during fine-tuning. However, looking at Section 5.1, we see that both the training and validation loss are very low. Also, with the number of epochs being a mere five, the prospect of overfitting is marginalised. However, we do observe that the validation rises from the penultimate epoch which indicates overfitting. Looking closer, we conclude that this change is minimal: 0.0005. Finally, the use of six different models in the benchmark reduces the risk of overfitting to a single task.

6.1.2 Job Shop Scheduling Dataset

As anticipated, the base Phi-2 model yielded poor results. Given that the base model is a raw, untuned model, it is natural for it to hallucinate and endlessly generate new tokens. This was observed in the predictions it generated. However, by fine-tuning the model, training it on this specific task, the scores increased dramatically, at least in a 0-shot setting, suggesting a novel understanding of the problem and how to solve it, even though it was not able to correctly complete any of the problems. Explanations for this fact include that the size of the problem instances was perhaps too big and complex for a small language model to actually solve, or that a bigger, more complex model might be able to more comprehensively model the relationship between the problem and solution. Moreover, the fine-tuned model showed a suboptimal understanding of the problem when using a 2-shot setting, performing in line with the base model, which possibly is an indication of overfitting to the exact format used during training. Finally, we observe that the average makespan difference greatly diminishes for the fine-tuned model which also suggests that it has gathered knowledge about the problem.

Furthermore, we also observe that the validation loss neatly follows the training loss across all epochs during fine-tuning. Generally, this is a sign that the model fits well to unseen data. However, both the training loss and validation loss stayed relatively high during the entire training, continuously remaining above 40%. This suggests that the problem itself might be too complex to model. Looking at the training and validation loss of the fine-tuned Phi-2 model on the OptiGuide benchmark datasets, we observe that there is a substantial difference. For this task, the loss is significantly higher, indicating poor convergence towards an optimal weight space minimum.

Usually, since the nature of language models lies in language, they are not developed to solve complex optimisation problems. The Phi-2 pre-trained base model has predominantly only been trained on textbook like data, solving common physics, maths and logical problems, as well as generating Python code (Microsoft, 2024). Despite the Phi-2 model's impressive nat-

ural language capabilities and its innovative approach to problem-solving, it could benefit from additional or improved training and potentially better data for further enhancement to more fluently and accurately solve optimisation problems. Another approach to solve optimisation problems like these could be to utilise reinforcement learning.

Transformer based language models have demonstrated remarkable proficiency across various domains. However, because of the immense parameter size of the models implicating immense complexity of the system, it is notably difficult to explain the behaviour of the model. Although it is possible to visualise and observe all 2.7 billion parameters during inference it is certainly not viable. Ultimately, because of this, the model can be seen as a black box. In order to fully trust models and employ them, explainability is of utmost concern. This limits our ability to draw conclusions about the model – especially when fine-tuned.

6.2 Fine-tuning SLMs

Elaborating further on the prospect of fine-tuning SLMs, we consider how they stand up against more popular LLMs. For instance, the most used series of LLMs, GPT, has a steadily increasing parameter size. The latest iteration in the series, GPT4, is estimated to have 1.76 trillion parameters. To fine-tune such a model would be far more costly than the fine-tuning we perform for this thesis. While this may be a viable option for the largest companies, SLMs such as the Phi models provide a much more reasonable choice for most organisations and individuals. As our results show that fine-tuning these models increases their performance on specific tasks, organisations can seek to do the same for their specialised tasks.

The development of SLMs, like LLMs, is all but stagnant. As Microsoft released the latest iteration in the Phi series, Phi-3, the performance of smaller models cut the gap to larger models even further. Phi-3 has a similar performance to GPT3.5, a model with a parameter size of around 175 billion. That is around 46 times more than the 3.8 billion of Phi-3-mini. With similar performance already from the base-model, fine-tuning at a relatively low cost can be a powerful tool for organisations performing specific tasks. Simultaneously, the pace of development of fine-tuning techniques is not any less rapid. Modern approaches, limiting the memory requirements and thus cost, provide even more opportunities to perform fine-tuning on SLMs. Furthermore, Phi models are not the only ones in their class, facing competition from models such as Mistral 7B, Llama 3 8B and Gemma 7B. Combined, these conditions provide a promising platform for further development and usage of language models to solve complex tasks at a reduced cost.

Finally, we note that fine-tuning a Phi-2 model has potential in a specific domain such as supply chain optimisation. SLMs have a good ability to adapt to and learn contextual specifics of a given domain. The results of our work apply not only to supply chain related tasks, but show that the methods applied can be useful in a supply chain context. Fine-tuned SLMs can be used by either supply chain managers or engineers engaged in supply chain optimisation.

6.3 Considerations and Limitations

Research in modern computer science requires thoughtful considerations. In a time of growing generative AI capacity, ethical considerations must be made. Modern models can produce toxic or inappropriate content, all while requiring massive amounts of computational power. We consider and reflect on our position as researchers in this domain.

6.3.1 Resources

Training language models requires substantial computational resources. Pre-training an LLM requires GPUs in the hundreds. For SLMs like the Phi models, several GPUs are still needed and training typically lasts for days. When fine-tuning an SLM as done in this project, GPU resources are also needed. These resources are sought-after and the fine-tuning process is expensive, both in terms of computation and in terms of cost. The GPU resource utilised for fine-tuning is detailed in Section 4.3.2. While techniques such as quantisation, parallelism and mixed precision training can be – and in our research are – employed to ease computational burden, the fact of the matter is that working with state-of-the-art language models today, whether large or small, requires expensive resources. Smaller models and the above mentioned techniques are steps on the way to democratising the development and usage of language models. Still, it is important to recognise the privileged position in which we are, to be able to work with these resources, as our research is fully dependent on it.

6.3.2 Ethical Considerations

The advent of Transformer based language models has ushered in a new era of possibilities within artificial intelligence. However, this recent progress is not without its ethical and environmental concerns. The immense computational power required to train these models is stunning, consuming vast amounts of energy, which ultimately raises sustainability concerns. Furthermore, the cost associated with developing these models presents an obstacle, since hugely expensive and specialised hardware is required, possibly limiting access to a select few who can afford them. While smaller models like the Phi models, and especially the new Phi-3, theoretically can run on smaller machines such as laptops and smart mobiles (Abdin et al., 2024), they still need large amounts of energy to be created. Despite these facts, there is hope that these technologies could contribute positively to combating climate change. Yet, the paradox remains: the models designed to help us may also be contributing to the problem, adding to the need of responsible innovation and use of such powerful technologies. Finally, it is therefore important to not only focus on the technological achievement in developing tools like these, but also reminisce on the consequences and possibilities for misuse they create.

Chapter 7
Conclusions

This chapter presents the conclusions from the results, presented in Chapter 5 and the following discussion, outlined in Chapter 6 in this paper. Section 7.1 presents the main conclusions drawn, while Section 7.2 lays forward potential future work using the results in this paper as a foundation.

7.1 Main Conclusions

In this thesis we research the performance and potential use of an SLM fine-tuned for specific tasks. We look closer at the Phi series of models and in particular we utilise Phi-2 for fine-tuning on supply chain optimisation tasks. We evaluate the performance of the base Phi-2 model and fine-tuned versions on two distinct tasks within supply chain optimisation, using two different datasets. The results show that fine-tuning increases the performance of Phi-2 vastly using the pre-defined metrics BLEU and ROUGE. For the code generation for what-if analysis task, the performance increases with 97 percentage points in a 0-shot setting when evaluated with BLEU, and increases with 92 percentage points in a 0-shot setting when evaluated with ROUGE. For the JSSP task, the performance increases with 30 percentage points in a 0-shot setting when evaluated with BLEU, and increases with 22 percentage points in a 0-shot setting when evaluated with ROUGE. While complex problems require thorough evaluation, these metrics indicate a substantial jump in model performance when fine-tuned. Importantly, while the base Phi-2 model suffers from hallucination and verbosity, the fine-tuned models exhibit improvement in task-specific coherence.

Furthermore, fine-tuning an SLM such as Phi-2 is viable considering the resource requirements. Using modern memory optimisation techniques like DeepSpeed and LoRA, Phi-2 can be fine-tuned using one single GPU. While we do not deliberately measure memory usage during fine-tuning, we use one NVIDIA A100 GPU during training which is more than sufficient for swiftly fine-tuning a Phi-2 model. To fine-tune larger models in the LLM class,

a substantial increase in resources would be required. In conclusion, organisations can leverage powerful capabilities of fine-tuned SLMs for a fraction of the cost of LLM fine-tuning.

With this in mind, we look back upon the three research questions we define in Section 1.1.1.

1. How well can a small language model perform when fine-tuned on specific tasks in supply chain optimisation?

    - A smaller model such as Phi-2 can to some extent be fine-tuned to solve tasks of different complexity. A fine-tuned model can correctly generate new Python code in what-if analysis within supply chain optimisation. On the other hand, a model fine-tuned on solving job shop scheduling tasks struggles to generate correct solutions. However, it still shows promising results, displaying novel knowledge of how to solve the problem at hand. The results indicate that a small model can be fine-tuned to tackle diverse tasks in supply chain optimisation.

2. To what extent does fine-tuning a small language model outperform prompt learning for its pre-trained base model?

    - Fine-tuned Phi-2 models greatly outperform the pre-trained base model using prompt learning on two distinct tasks. For the code generation task, the fine-tuned model in a 0-shot setting strongly outperforms the Baseline, which suffers from hallucination and verbosity. For the job shop scheduling problem, the fine-tuned model in a 0-shot setting strongly outperformed the same model with a 2-shot setting. In both settings, the fine-tuned model outperforms the Baseline.

3. Does the small parameter size and resource efficiency of small language models impact their practical viability for fine-tuning on specific tasks?

    - The small Phi-2 model can be fine-tuned to increase performance and solve applied tasks in a specific domain using relatively limited resources. Using state-of-the-art memory optimisation techniques, Phi-2 can be fine-tuned without losing performance. This makes fine-tuning small language models a viable option in times of ever-growing model sizes and resource requirements.

7.2 Future Work

To extend our research, we propose a number of ideas for future work. First and foremost, as of writing, a new model in the Phi series has been released – Phi-3 (Abdin et al., 2024), unveiling new possibilities for SLMs in the area of supply chain optimisation. Our research in this thesis can therefore naturally be extended with Phi-3. Furthermore, the use and fine-tuning of other SLMs such as Llama 3 8B (Meta, 2024), Gemma 7B (Mesnard et al., 2024) or Mistral 7B (Jiang et al., 2023) in this area is also of great interest. The use of different models in the area of supply chain optimisation applies greatly in the case of OptiGuide. However, research on optimisation problems such as JSSP might benefit more from another approach, utilising reinforcement learning techniques for example.

Additionally, new advancements are constantly being made in both the fields of machine learning and natural language processing. The pace of development in the field of memory optimisation techniques is rapid. Coupled with hardware advancements, especially in GPUs, the process of fine-tuning has potential to become more available and efficient. The same goes for the advancements of language models, both large and small. Remarkable advancements are made at an unprecedented pace.

Finally, while this thesis covers two distinct cases in the area of supply chain optimisation, our methodology can naturally be extended to other tasks in the same domain. Fine-tuning an SLM to solve different types of tasks in supply chain optimisation can be done given access to structured data. For example, our research can possibly be applied to tasks within risk management or demand forecasting.

References

Abdin, M., Jacobs, S. A., Awan, A. A., Aneja, J., Awadallah, A., Awadalla, H., Bach, N., Bahree, A., Bakhtiari, A., Behl, H., Benhaim, A., Bilenko, M., Bjorck, J., Bubeck, S., Cai, M., Mendes, C. C. T., Chen, W., Chaudhary, V., Chopra, P., Giorno, A. D., de Rosa, G., Dixon, M., Eldan, R., Iter, D., Garg, A., Goswami, A., Gunasekar, S., Haider, E., Hao, J., Hewett, R. J., Huynh, J., Javaheripi, M., Jin, X., Kauffmann, P., Karampatziakis, N., Kim, D., Khademi, M., Kurilenko, L., Lee, J. R., Lee, Y. T., Li, Y., Liang, C., Liu, W., Lin, E., Lin, Z., Madan, P., Mitra, A., Modi, H., Nguyen, A., Norick, B., Patra, B., Perez-Becker, D., Portet, T., Pryzant, R., Qin, H., Radmilac, M., Rosset, C., Roy, S., Ruwase, O., Saarikivi, O., Saied, A., Salim, A., Santacroce, M., Shah, S., Shang, N., Sharma, H., Song, X., Tanaka, M., Wang, X., Ward, R., Wang, G., Witte, P., Wyatt, M., Xu, C., Xu, J., Yadav, S., Yang, F., Yang, Z., Yu, D., Zhang, C., Zhang, C., Zhang, J., Zhang, L. L., Zhang, Y., Zhang, Y., Zhang, Y., and Zhou, X. (2024). Phi-3 technical report: A highly capable language model locally on your phone. arXiv:2404.14129.

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. (2021). Program synthesis with large language models. arXiv:2108.07732.

Bläckberg, H. (2024). Optimizing Soak Test Reviews: A Comparative Study of Deep Learning Architectures. Student Paper.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. arXiv:2005.14165.

Chen, B., Zhang, Z., Langrené, N., and Zhu, S. (2023a). Unleashing the potential of prompt engineering in large language models: a comprehensive review. arXiv:2310.14735.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y.,

Joseph, N., Brockman, G., et al. (2021). Evaluating large language models trained on code. arXiv:2107.03374.

Chen, X., Wang, Y., Du, Y., Hassoun, S., and Liu, L.-P. (2023b). On separate normalization in self-supervised transformers. arXiv:2309.12931.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. (2021). Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*. arXiv:2110.14168.

Dettmers, T., Pagnoni, A., Holtzman, A., and Zettlemoyer, L. (2023). Qlora: Efficient finetuning of quantized llms. arXiv:2305.14314.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Floridi, L. and Chiriatti, M. (2020). Gpt-3: Its nature, scope, limits, and consequences. *Minds Mach.*, 30(4):681–694.

Frieske, R. and Shi, B. E. (2024). Hallucinations in neural automatic speech recognition: Identifying errors and hallucinatory models. arXiv:2401.01572.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Gunasekar, S., Zhang, Y., Aneja, J., Mendes, C. C. T., Giorno, A. D., Gopi, S., Javaheripi, M., Kauffmann, P., de Rosa, G., Saarikivi, O., Salim, A., Shah, S., Behl, H. S., Wang, X., Bubeck, S., Eldan, R., Kalai, A. T., Lee, Y. T., and Li, Y. (2023). Textbooks are all you need. arXiv:2306.11644.

Gurobi (2024). Python examples. `https://www.gurobi.com/documentation/current/examples/python_examples.html`.

Haque, A. and Ghani, S. (2022). The storyteller: Computer vision driven context and content generation system. Available at SSRN.

Hu, E. J., Shen, Y., Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., and Chen, W. (2021). Lora: Low-rank adaptation of large language models. arXiv:2016.09685.

HuggingFace (2024). bitsandbytes. `https://huggingface.co/docs/bitsandbytes/main/en/index`.

Jiang, A. Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D. S., de las Casas, D., Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L. R., Lachaux, M.-A., Stock, P., Scao, T. L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W. E. (2023). Mistral 7b. arXiv:2310.06825.

Li, B., Mellou, K., Zhang, B., Pathuri, J., and Menache, I. (2023a). Large language models for supply chain optimization. arXiv:2307.03875.

Li, Y., Bubeck, S., Eldan, R., Giorno, A. D., Gunasekar, S., and Lee, Y. T. (2023b). Textbooks are all you need ii: phi-1.5 technical report. arXiv:2309.05463.

Lin, C.-Y. (2004). ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2021). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. arXiv:2107.13586.

Loshchilov, I. and Hutter, F. (2017). Fixing weight decay regularization in adam. arXiv:1711.05101.

McDonald, J., Li, B., Frey, N., Tiwari, D., Gadepally, V., and Samsi, S. (2022). Great power, great responsibility: Recommendations for reducing energy for training language models. In *Findings of the Association for Computational Linguistics: NAACL 2022*. Association for Computational Linguistics.

Mesnard, T., Hardin, C., Dadashi, R., Bhupatiraju, S., Pathak, S., Sifre, L., Rivière, M., Kale, M. S., Love, J., Tafti, P., Hussenot, L., Sessa, P. G., Chowdhery, A., Roberts, A., Barua, A., Botev, A., Castro-Ros, A., Slone, A., Héliou, A., Tacchetti, A., Bulanova, A., Paterson, A., Tsai, B., Shahriari, B., Lan, C. L., Choquette-Choo, C. A., Crepy, C., Cer, D., Ippolito, D., Reid, D., Buchatskaya, E., Ni, E., Noland, E., Yan, G., Tucker, G., Muraru, G.-C., Rozhdestvenskiy, G., Michalewski, H., Tenney, I., Grishchenko, I., Austin, J., Keeling, J., Labanowski, J., Lespiau, J.-B., Stanway, J., Brennan, J., Chen, J., Ferret, J., Chiu, J., Mao-Jones, J., Lee, K., Yu, K., Millican, K., Sjoesund, L. L., Lee, L., Dixon, L., Reid, M., Mikuła, M., Wirth, M., Sharman, M., Chinaev, N., Thain, N., Bachem, O., Chang, O., Wahltinez, O., Bailey, P., Michel, P., Yotov, P., Chaabouni, R., Comanescu, R., Jana, R., Anil, R., McIlroy, R., Liu, R., Mullins, R., Smith, S. L., Borgeaud, S., Girgin, S., Douglas, S., Pandya, S., Shakeri, S., De, S., Klimenko, T., Hennigan, T., Feinberg, V., Stokowiec, W., hui Chen, Y., Ahmed, Z., Gong, Z., Warkentin, T., Peran, L., Giang, M., Farabet, C., Vinyals, O., Dean, J., Kavukcuoglu, K., Hassabis, D., Ghahramani, Z., Eck, D., Barral, J., Pereira, F., Collins, E., Joulin, A., Fiedel, N., Senter, E., Andreev, A., and Kenealy, K. (2024). Gemma: Open models based on gemini research and technology. arXiv:2403.08295.

Meta (2024). Build the future of ai with meta llama 3. `https://llama.meta.com/llama3/`.

Microsoft (2023). Phi-2: The surprising power of small language models. `https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/`.

Microsoft (2024). Azure ai studio: Microsoft phi-2. `https://ai.azure.com/explore/models/microsoft-phi-2/version/4/registry/azureml-msr`.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. arXiv:1301.3781.

Miller, C. E., Tucker, A. W., and Zemlin, R. A. (1960). Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329.

Nugues, P. (2016). *Language Processing with Perl and Prolog: Theories, Implementation, and Application*. Cognitive Technologies. Springer Berlin Heidelberg.

Ohno, T. (1988). *Toyota Production System: Beyond Large-Scale Production*. Taylor & Francis.

Oliveira, E. e. and Pereira, T. (2023). A new generation? a discussion on deep generative models in supply chains. In Alfnes, E., Romsdal, A., Strandhagen, J. O., von Cieminski, G., and Romero, D., editors, *Advances in Production Management Systems. Production Management Systems for Responsible Manufacturing, Service, and Logistics Futures*, pages 444–457, Cham. Springer Nature Switzerland.

OpenAI (2024). Fine-tuning. `https://platform.openai.com/docs/guides/fine-tuning`.

Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In Isabelle, P., Charniak, E., and Lin, D., editors, *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318, Philadelphia, Pennsylvania, USA. Association for Computational Linguistics.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Pope, R., Douglas, S., Chowdhery, A., Devlin, J., Bradbury, J., Levskaya, A., Heek, J., Xiao, K., Agrawal, S., and Dean, J. (2022). Efficiently scaling transformer inference. arXiv:2211.05102.

Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training. `https://cdn.openai.com/research-covers/languageunsupervised/language_understanding_paper.pdf`.

Rajbhandari, S., Rasley, J., Ruwase, O., and He, Y. (2020). Zero: Memory optimizations toward training trillion parameter models. arXiv:1910.02054.

Ruder, S. (2016). An overview of gradient descent optimization algorithms. arXiv:1609.04747.

Russell, S. and Norvig, P. (2020). *Artificial Intelligence: A Modern Approach*. Pearson series in artificial intelligence. Pearson.

Samsi, S., Zhao, D., McDonald, J., Li, B., Michaleas, A., Jones, M., Bergeron, W., Kepner, J., Tiwari, D., and Gadepally, V. (2023). From words to watts: Benchmarking the energy costs of large language model inference. arXiv:2310.03003.

Sharir, O., Peleg, B., and Shoham, Y. (2020). The cost of training nlp models: A concise overview. arXiv:2004.08900.

Shavaki, F. H. and Ghahnavieh, A. E. (2022). Applications of deep learning into supply chain management: a systematic literature review and a framework for future research. *Artificial Intelligence Review*, 56(5):4447–4489.

Taillard, E. D. (1993). Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278?285.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. arXiv:2307.03875.

Watson, M., Lewis, S., Cacioppi, P., and Jayaraman, J. (2012). *Supply Chain Network Design: Applying Optimization and Analytics to the Global Supply Chain*. FT Press Operations Management. Pearson Education.

Zhou, Z., Li, L., Chen, X., and Li, A. (2023). Mini-giants: "small" language models and open source win-win. arXiv:2307.08189.

Appendices

Appendix A
Abbreviations

AI ..... Artificial Intelligence

BERT ..... Bidirectional Encoder Representations from Transformers

BLEU ..... Bilingual Evaluation Understudy

BPE ..... Byte-Pair Encoding

CPU ..... Central Processing Unit

GPT ..... Generative Pre-trained Transformer

GPU ..... Graphics Processing Unit

JIT ..... Just-in-Time

JSSP ..... Job Shop Scheduling Problem

LLM ..... Large Language Model

LCS ..... Longest Common Subsequence

LP ..... Linear Programming

LoRA ..... Low-Rank Adaptation

MIP ..... Mixed Integer Programming

MBPP ..... Mostly Basic Python Programming

ML ..... Machine Learning

MLM ..... Masked Language Modelling

MLP . . . . . Multi-Layer Perceptron

NLP . . . . . Natural Language Processing

NVMe . . . . . Non-Volatile Memory express

PEFT . . . . . Parameter-Efficient Fine-Tuning

QA . . . . . Question-Answer

QLoRA . . . . . Quantised Low-Rank Adaptation

ReLU . . . . . Rectified Linear Unit

RNN . . . . . Recurrent Neural Network

ROUGE . . . . . Recall-Oriented Understudy for Gisting Evaluation

SOTA . . . . . State-of-the-art

SGD . . . . . Stochastic Gradient Descent

SLM . . . . . Small Language Model

SSD . . . . . Solid-State Drive

TSP . . . . . Travelling Salesman Problem

VM . . . . . Virtual Machine

ZeRO . . . . . Zero Redundancy Optimiser

Appendix B
A Small Note on Cover Art

The cover page of this master's thesis features a detail of an Aeonium tabuliforme from Trädgårdsföreningen, Gothenburg, Sweden. The photograph is taken by Max Ronnersjö and has been resized. It is used under the CC BY-SA 3.0 license.

The Aeonium tabuliforme has a multiple spiral arrangement, known as parastichy, which seemingly follows the golden ratio, seen appearing in phyllotaxis. The golden ratio, in mathematics, is defined as

$$\frac{a+b}{a} = \frac{a}{b} = \phi \tag{B.1}$$

where a and b are two quantities. Equation B.1 describes the case where the ratio of a and b is the same as the ratio of their sum to the larger quantity. In this case, a is in golden ratio to b. The golden ratio is denoted as the Greek letter ϕ (phi), a name it shares with the family of small language models researched in this thesis, *Phi*.

Appendix C
Division of Work

Area	A. Beke	T. Zitouni
Research	50%	50%
Implementation	50%	50%
Data gathering and processing	50%	50%
Microsoft Azure fine-tuning engineering	25%	75%
Microsoft Azure endpoint deployment	75%	25%
Graphical design	80%	20%
Coordination between LTH and Microsoft	20%	80%
Writing the paper	50%	50%

Table C.1: Contribution of the authors to the different segments of the effort behind this master's thesis.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2024-05-30

EXAMENSARBETE Fine-tuning Phi models for Informed Decision Support in Supply Chain Optimisation
STUDENTER Axel Beke, Théodore Zitouni
HANDLEDARE Pierre Nugues (LTH), Bahram Zarrin (Microsoft)
EXAMINATOR Jacek Malec (LTH)

Phi-2: en liten språkmodell med stor kapacitet

POPULÄRVETENSKAPLIG SAMMANFATTNING **Axel Beke, Théodore Zitouni**

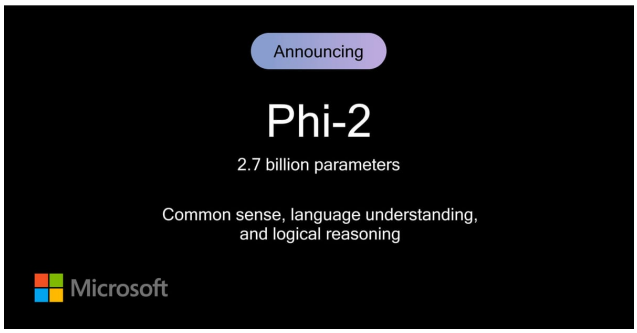
Distributionskedjeoptimering är ett brett och komplicerat område. Utvecklingen av Transformerbaserade språkmodeller öppnat upp för ny forskning inom det här området. Arbetet utforskar användandet av små språkmodeller inom två områden: kodgenerering för what-if analys och att lösa job shop scheduling problemet.

Distributionskedjeoptimering är ett viktigt område i industrin. Med detta kan organisationer ta bättre och mer välinformerade beslut och öka robustheten i kedjor. Efter den senaste tidens utveckling av Transformerbaserade språkmodeller har nya möjligheter för forskning som applicerar dessa uppstått.

Vi undersöker möjligheten att finjustera Phi-modeller för att bättre lösa två skilda uppgifter inom distributionskedjeoptimering: kodgenerering för what-if analys och att lösa job shop scheduling problemet. What-if analys tillåter företag att analysera olika scenarion som kan uppstå. Job shop schedulingq är ett optimimeringsproblem som går ut på att schemalägga olika jobb till olika maskiner.

Vi applicerar Phi-2, en liten språkmodell från Microsoft, för att lösa dessa problem. Medan språkmodeller kontinuerligt har växt i parameterstorlek och komplexitet, har så även kraven på minnesresurser. Denna utveckling har lett till ett forskningsintresse i att konstruera mindre språkmodeller som alltjämt har en hög resoneringsförmåga.

Finjustering av språkmodeller har ett övergripande mål, att bli bättre på ett specifikt område som modellen inte tränats på i grundträningen.



Genom att träna om ett mindre antal av lagerna, kan språkmodeller bli domänexperter utan att förlora förmågan att resonera. Moderna metoder med låga krav på minnesresurser har medfört att finjustering av språkmodeller kan vara av intresse för organisationer för varierade problem.

Våra resultat visar att Phi-2 kan finjusteras för ökad prestation på kontextualiserad kodgenerering för what-if analys inom distributionskedjor, med en BLEU-poäng – en vanligt förekommande evalueringsmetod – på 98.6%. Job shop scheduling problemet är mer komplext, men den finjusterade modellen visar på en nyskapad förståelse en BLEU-poäng på 21.7% som bäst. Finjustering av små språkmodeller har stor potential inom distributionskedjeoptimering.