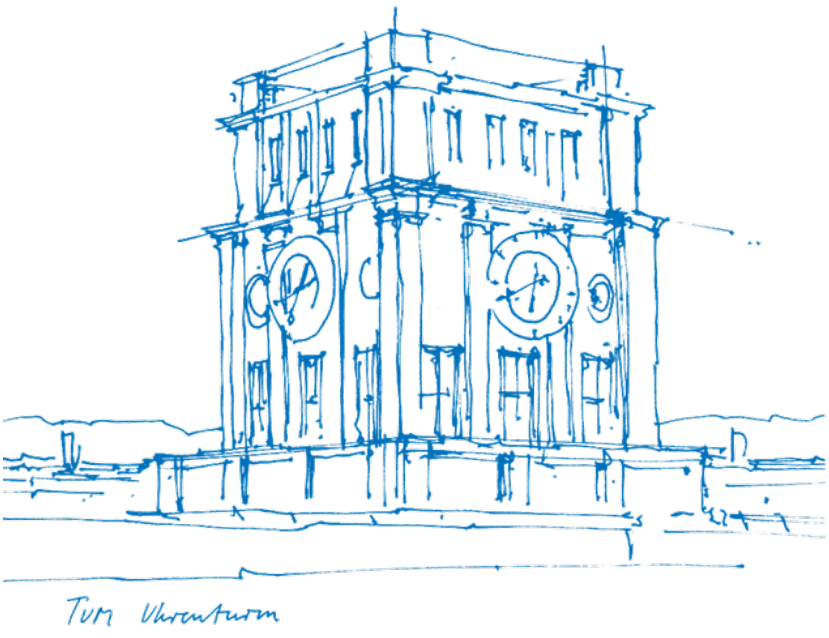
Chair of Information Systems and Business Process Management (i17)
Department of Computer Science
TUM School of Computation, Information and Technology
Technical University of Munich

Master's Thesis in Information Systems

Furkan Gürbüz

Fine-Tuning Large Language Model with Custom Dataset for Ansible Code Generation



Chair of Information Systems and Business Process Management (i17)
Department of Computer Science
TUM School of Computation, Information and Technology
Technical University of Munich

Master's Thesis in Information Systems

Furkan Gürbüz

Fine-Tuning Large Language Model with Custom Dataset for Ansible Code Generation

Feinabstimmung eines großen Sprachmodells mit benutzerdefiniertem Datensatz zur Code-Generierung in Ansible

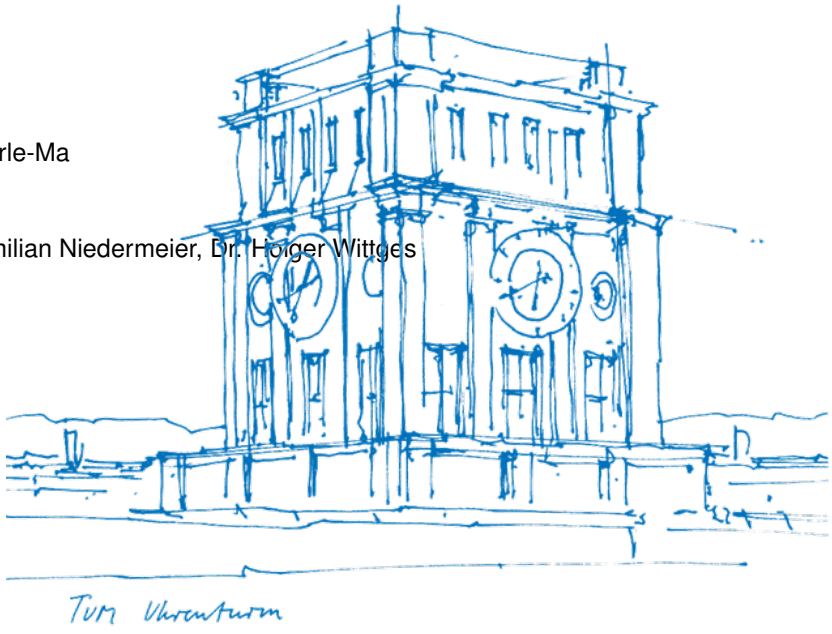
Thesis for the Attainment of the Degree
Master of Science

at the TUM School of Computation, Information and Technology,
Department of Computer Science,
Chair of Information Systems and Business Process Management (i17)

Examiner
Prof. Dr. Stefanie Rinderle-Ma

Supervised by
Thomas Teubner, Maximilian Niedermeier, Dr. Holger Wittges

Submitted on
01.06.2025



Declaration of Academic Integrity

I confirm that this bachelor's thesis is my own work and I have documented all sources and material used.

I am aware that the thesis in digital form can be examined for the use of unauthorized aid and in order to determine whether the thesis as a whole or parts incorporated in it may be deemed as plagiarism. For the comparison of my work with existing sources I agree that it shall be entered in a database where it shall also remain after examination, to enable comparison with future theses submitted. Further rights of reproduction and usage, however, are not granted here.

This thesis was not previously presented to another examination board and has not been published.

Garching, 01.06.2025 Furkan Gürbüz

Abstract

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Keywords: *Include three to five words, phrases, or acronyms as keywords.*

Contents

List of Tables

List of Figures

Introduction

Motivation

In the era of digital transformation, the ability to automate IT operations has become a fundamental driver of organizational efficiency and scalability. Modern enterprises and research institutions are increasingly adopting rapid development cycles that demand frequent updates and enhancements (Gupta et al., 2019). This continuous evolution requires efficient system management, making the automation of deployments and configurations a critical aspect of DevOps practices (Bass et al., 2015).

Furthermore, Enterprise Resource Planning (ERP) systems are crucial to achieve efficiency and scalability by integrating and streamlining essential business functions (Poston and Grabski, 2000). Among these, SAP stands out as a leading ERP solution, continuously evolving its capabilities since its inception in Germany (Klaus et al., 2000). Educational institutions, such as the SAP University Competence Center at the Technical University of Munich, play a crucial role in providing training and equipping future professionals with the knowledge and skills required to operate ERP systems proficiently (UCC, 2024).

In addition to educating future professionals, the SAP UCC also offers hosting of SAP solutions with robust backup and recovery services. The center utilizes an IT Automation software to install and configure SAP systems (UCC, 2024). The software plays a critical role in automating administrative tasks, such as configuration management and application deployment, through the implementation of playbooks. (Ansible, 2024). Manually developing these playbooks is inherently complex and time consuming, requiring a nuanced understanding of the implementation syntax and the specific requirements of SAP systems (Geerling, 2015). This complexity poses a substantial challenge for organizations that want to quickly scale operations or fully embrace automation.

Consequently, there is a substantial demand for code generation and support in implementing Ansible playbooks. As it stands, the academic chair does not possess adequate tools for code generation support in Ansible, and existing models do not meet the requirements satisfactorily. By elevating automation capabilities, IT professionals will be well-equipped to optimize operations.

This initiative aligns with trends with focus on enhancing operational efficiency, reducing costs, and accelerating deployment timelines.

Research Questions

This thesis will cover multiple steps involved in fine-tuning a large language model. Firstly, we will create a custom dataset required for the fine-tuning process, ensuring consistency and tailoring the model to our specific use case. Subsequently, we will fine-tune the large language model, a process that requires significant computational resources and coding. Finally, we will conduct a testing and evaluation phase to assess the fine-tuned model.

After implementing the fine-tuned Large Language Model, the deployment process of the SAP UCC will be enhanced due to faster implementation of Ansible playbooks facilitated with the optimized model.

Research Question 1: What methodology can be used to gather and prepare a custom dataset for fine-tuning large language models?

Methodology:

In the context of this research question, the focus is on the creation of a custom dataset, which will be used for fine-tuning the large language model. We employ a Question Answering dataset format, utilizing specific tools to gather our dataset. Following this, we will clean and preprocess the data to eliminate noise and ensure consistency in data quality and format. This step is crucial to optimizing the data set for model training. The data needs to be partitioned into different types of set. This will ensure robustness and reliability of the trained model for our use case.

Expected results:

We have created a dataset that will be cleaned and prepared for fine-tuning the large language model.

Research Question 2: What steps are involved in implementing the fine-tuning process for the large language model?

Methodology:

In the context of this research question, the focus is on the fine-tuning process with the created dataset from RQ1. This research question will focus on the implementation and execution of fine-tuning strategies.

Expected results:

We have tuned the large language model, that is suitable for our use case.

Research Question 3: To what extend does the fine-tuned large language model meet the requirements in terms of performance, accuracy, and applicability?

Methodology:

Concerning this research question the focus is on testing the model performance. Consequently, we test our model's performance with professionals by ensuring the usability, accuracy and functionality.

Expected results:

We have tested our model and give insights on the performance of our fine tuned large language model in the context of code generation.

Contribution

The goal of this thesis is to optimize the implementation of Ansible playbooks by using a fine-tuned large-language model. By generating Ansible code specifically tailored for SAP systems, this thesis aims to reduce the time and expertise required for playbook development, making the process more efficient and accessible for both developers and system administrators. This will enable them to focus on higher-level tasks and strategic decision-making. This approach will lead to more consistent, error-free configurations and contribute to the streamlined management of complex IT systems.

In order to achieve this objective, the thesis will involve fine-tuning a large language model specifically for Ansible code generation. The process will begin with the creation of a custom dataset containing Ansible code specifically designed to meet the unique requirements of SAP system configurations. Once the dataset is created, it will undergo preparation and pre-processing to ensure consistency and remove noise. The data set will then be used to train the model, enabling it to generate context-specific high-quality Ansible playbooks (Howard and Ruder, 2018). Using this custom data set, the model will learn to produce accurate and efficient Ansible YAML code.

Afterwards, the fine-tuning process follows several key steps, beginning with the selection of a pre-trained model. The model parameters will be fine-tuned by training it on the custom dataset, adjusting the model to generate code that is syntactically correct and functionally effective. Fine-tuning iterations will involve gradually adjusting the hyperparameters and retraining the model to

optimize its performance. Upon completion of the fine-tuning process, the model's performance will be tested using a separate validation set to assess its accuracy and effectiveness.

The expected outcomes include not only time and cost savings, but also the ability to scale automation across more complex IT environments. Through the integration of advanced machine learning techniques, this research will contribute to the ongoing evolution of IT automation practices.

Methodology

Design Science Research (DSR) is a well-established methodology in the field of information systems researchIts main focus is to create and evaluate practical solutions—called artifacts—that address real-world problems. First introduced by Hevner et al., DSR highlights the importance of not only building innovative solutions but also thoroughly testing them to bridge the gap between theory and practice (Hevner et al., 2010). The key Components of Design Science Research are Stakeholders and Artifacts. Stakeholders within DSR are divided into different roles:

- **Researchers:** The individuals who conduct the research and implement the artifacts.
- **Practitioners and Users:** The end-users and organizations that will benefit from using the artifacts.
- **Reviewers and Evaluators:** The experts who evaluate the functionality and impact of the artifacts.
- **Sponsors and Funding Bodies:** The organizations or institutions providing financial support for the research initiative.

Artifacts in DSR - as defined by Hevner et al. - include constructs, models, methods, and instantiations, all serving as outputs of the research process that are both innovative and rigorously evaluated (Hevner et al., 2010). Hevner et al. outline a systematic six-step process to conduct DSR (Hevner et al., 2010), which ensures both scientific rigor and practical relevance:

1. **Problem Identification and Motivation:** Clearly describe the research problem and justify its importance. Provide a detailed analysis to establish the relevance and urgency of the problem (Hevner et al., 2010.

2. **Define Objectives of a Solution:** Establish the criteria and benchmarks for an effective solution. Clearly define what constitutes success for the artifact being developed and how it differs or improves existing solutions (Hevner et al., 2010.
3. **Design and Development:** Develop the artifact using theoretical insights and design principles. Specify the components, functionalities and intended applications of the artifact (Hevner et al., 2010.
4. **Demonstration:** Apply the artifact in a controlled or real-world setting to prove its practical utility. Provide empirical evidence that the artifact can effectively address the identified problem (Hevner et al., 2010.
5. **Evaluation:** Assess the performance of the artifact against predefined objectives. Use qualitative and quantitative measures to assess the utility, efficacy, and quality of the artifact, gathering feedback from key stakeholders (Hevner et al., 2010.
6. **Communication:** Document the research process, findings, and implications. Disseminate results to both academic and practitioner communities to highlight contributions to theory and practical applications (Hevner et al., 2010).

In this thesis, Hevner's DSR principles will guide the research design. This includes structuring the problem definition and solution design process according to the DSR steps. Additionally, emphasizing the development and preliminary evaluation of a solution artifact, and involving stakeholders to validate the research outputs and ensure practical relevance. By integrating elements of Hevner's DSR, the research aims to maintain a structured, rigorous approach that enhances the significance, relevance, and applicability of the implemented solutions.

In our use case, stakeholders are defined as follows. The researchers consist of the thesis writer and the supervisor. The users include developers and system managers at the SAP UCC Chair. The reviewers and evaluators, a subgroup of the users, will be responsible for assessing the artifact. The funding body supporting this research is the Technical University of Munich.

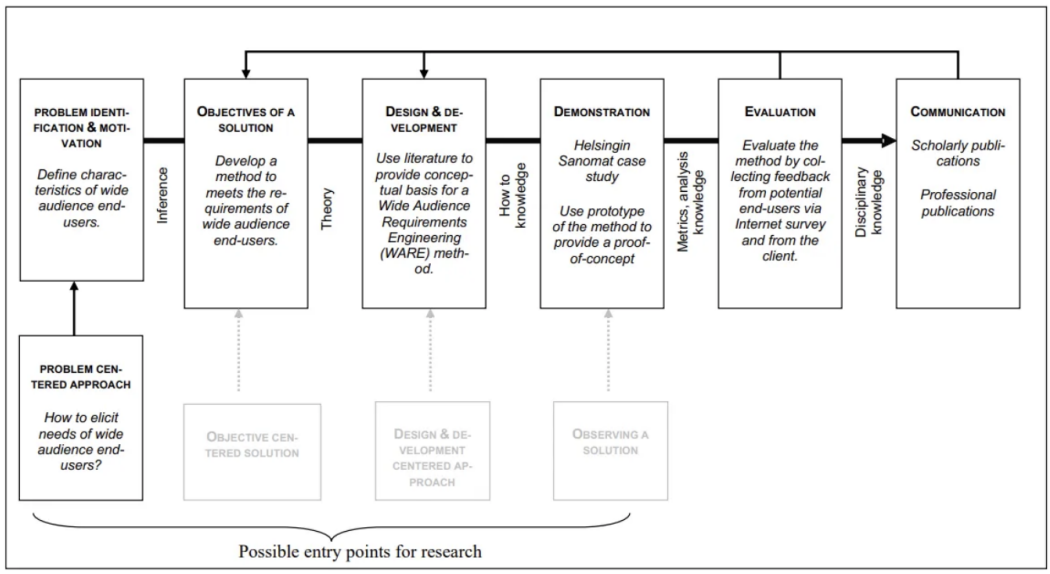
The primary artifact of this research is the fine-tuned large language model designed to generate improved Ansible code. The artifact aims to enhance the quality, efficiency, and accuracy of code generation to support developers and system managers in the SAP UCC Chair.

Evaluation

The evaluation will be conducted through multiple iterative cycles, based on the predefined DSR methodology as illustrated in Figure 3, where each iteration consists of two phases. One phase includes the model fine-tuning process with adjusted training configurations and training data. The second phase consists of the evaluation of the performance by predefined metrics which focus on the model's accuracy in generating code (functional correctness) and syntactical correctness.

After each iteration, the results will be evaluated to see whether the adjustments applied will positively affect the outcome. Once a satisfied performance level is reached or a further iteration seems redundant, the model will be submitted to another qualitative evaluation.

Figure 1
DSR process cycles



Overview of the iterative phases in DSR process (Peffers et al., 2007).

This final evaluation phase involves a group of experts with experience in the development and maintenance of Ansible playbooks in sap environments. Therefore, experts will analyze the generated code, along with the corresponding prompts. Feedback gathered from expert evaluations will be analyzed and documented. If critical issues or potential improvements are identified, additional iteration cycles will be considered to address these points. This whole process not only ensures an optimized and fine-tuned model but is also validated by experts, aligning the technical results with practical expectations.

Structure

This thesis will be structured into seven main chapters. First, an introduction into the conducting research will be given. Afterwards, we will discuss related work based on custom dataset creation and fine-tuning a large language model. Additionally, we will cover the solution design with focus on dataset creation and fine-tuning strategy. The implementation chapter will describe a detailed overview about tools, frameworks and programming languages needed for the development of the solution. Hereby, encountered challenges and problems will also be covered. The evaluation chapter will explain the experiment to evaluate the tuned model and additionally justify the metrics that are used for testing. Furthermore, a discussion chapter will interpret the results and link them to the research questions stated in the introduction. Hereby, implications of the results will be discussed for future research. Lastly, a conclusion with a summary of the key findings and research contributions will be presented. With this structure, the thesis aims to provide a clear and logical flow, guiding the reader from the initial research motivation and background, through the technical implementation and evaluation, to the final discussion and conclusion, ensuring that all relevant aspects of the research process are covered in a transparent and comprehensible way.

Related Work

Large language models (LLMs) have a significant impact on the field of computer science, introducing new possibilities across various domains. LLMs decreased the complexity in automation and improved workflows. In recent years, multiple studies show that LLMs create value in technical workflows by reducing the time spent on manual tasks (Brown et al., 2020; Radford et al., 2021). The increased popularity of these models is because of their flexibility. LLMs can generate code, they can assist developers by automating repetitive tasks, documenting workflows, and providing context-aware explanations for complex code fragments, thereby supporting both novice and experienced developers.

Hereby, it is important to discuss the advantages and challenges within the integration of LLMs into workflows. On the one hand, research has shown that LLMs can accelerate development cycles. enhance code quality, and minimize common errors (Chen et al., 2021). On the other hand, ensuring that the generated code is interpretable and trustworthy remains a crucial issue. Furthermore, data privacy concerns arise when proprietary or sensitive codebases are involved, and the need to fine-tune models for specialized domains introduces additional complexity (Bender et al., 2021).

Within the context of this research, LLMs are particularly promising for supporting the development and maintenance of Ansible playbooks in SAP environments. By fine-tuning a dedicated model, such as Phi-4, to the specific requirements and conventions of SAP infrastructure automation, the model can assist developers in multiple ways. For example, it can generate playbook templates, suggest improvements to existing code, automate routine configuration tasks, and provide explanatory comments for non-trivial sections of automation logic. These capabilities not only enhance productivity but also contribute to knowledge transfer, making it easier for new team members to understand and contribute to the automation process. Additionally, since the fine-tuning process incorporates domain-specific best practices, the model's recommendations are better aligned with the operational and technical constraints of SAP systems, ultimately improving the reliability and maintainability of the resulting playbooks (Hevner et al., 2010; Vaswani et al., 2017).

The use of LLMs to automate IT tasks has seen significant growth in recent years. These models have the potential to greatly enhance developer productivity, especially when applied in domain-

specific contexts. One of the prominent projects in this area is Ansible Lightspeed, developed by Sahoo et al. (2024). Ansible Lightspeed is a generative AI service specifically designed to generate Ansible YAML code. This service utilizes the IBM Watsonx Code Assistant for Red Hat Ansible (WCA-Ansible), a transformer-based decoder with 350 million parameters. The model was trained from the ground up on a diverse set of natural language, source code, and Ansible-specific data. By providing code recommendations based on natural language prompts, Ansible Lightspeed supports developers in streamlining their IT automation tasks. The effectiveness of the model was evaluated through interaction data from more than 10,000 users, achieving a notable 13.66 percent N-day user retention rate on day 30 (Sahoo et al., 2024). These figures demonstrate that a specialized model can achieve high acceptance rates among its users when deployed effectively.

In contrast to Ansible Lightspeed, our model is specifically tailored to automation tasks within SAP environments. While Ansible Lightspeed relies on general Ansible data, our model is trained on a dataset specifically curated for SAP-related tasks. This SAP-focused dataset is created using the Ansible Content Parser, a tool that meticulously extracts relevant content from existing SAP sources. Using this specialized data set, we anticipate that our model will offer more detailed and context-specific code recommendations that are perfectly suited to SAP environments.

There are several key differences between Ansible Lightspeed and our fine-tuned model within this research. First, the domain focus sets them apart. Ansible Lightspeed is designed for general IT automation tasks using Ansible, whereas our model is specifically targeted at automating SAP environments. This specialization ensures that our model can address the unique requirements and complexities associated with SAP systems. Secondly, the datasets used to train the models differ significantly. Ansible Lightspeed is trained on generic Ansible data, encompassing source code and natural language inputs from a wide range of domains. In contrast, our model leverages a custom dataset specifically designed for SAP environments. This allows us to create a dataset that compensates the unique requirements of SAP environments. Our model can provide code recommendations that are more accurate and more relevant by concentrating on this customized data, guaranteeing that they precisely match the requirements of SAP systems. Third, each model has a distinct user base. A flexible strategy that can be adjusted to different use cases is required because Ansible Lightspeed is made to accommodate a broad spectrum of developers that use Ansible for a variety of IT automation jobs. On the other hand, our strategy is designed for a niche market: customers that are particularly interested in using Ansible playbooks to automate SAP

systems. Finally, the training methodologies and the models themselves differ significantly. Ansible Lightspeed relies on the IBM Watsonx Code Assistant, which is designed for general-purpose code generation and leverages a broad dataset to ensure flexibility across various IT automation contexts. In contrast, our approach utilizes the Phi-4 model, which is fine-tuned specifically for SAP environments using a custom dataset. Existing code completion systems such as GitHub Copilot (GitHub, 2021), Tab9 (Tab9, 2018), Replit (Replit, 2016), and Amazon Code Whisperer (Amazon, 2023) represent more generic approaches to source code generation. These tools support a variety of programming languages and tasks, demonstrating the versatility of LLMs in software development. However, studies have shown that specialized models, like our SAP-focused model and Ansible Lightspeed, can achieve higher acceptance rates and greater user satisfaction by tailoring optimizations to specific domains (Ziegler et al., 2022; Pujar et al., 2023).

We believe that our fine-tuned model, specifically focused on SAP scenarios, will deliver comparable or even superior results relative to existing generic models. By precisely addressing the needs of developers working in SAP environments, our model has the potential to significantly enhance their efficiency and productivity. Ultimately, this could set a new benchmark for future developments in domain-specific AI-driven code generation tools.

Solution Design

Solution Design: Architecture, rationale, specifications, algorithms, mockups

Distinction of small and large language models

Artificial intelligence now relies heavily on language models, especially in natural language processing (NLP). The number of parameters, computational complexity, and data efficiency set Large Language Models (LLMs) apart. LLMs show gains in thinking, problem-solving, and generalization across a range of tasks as they grow. Even though models with more than 100 billion parameters, like GPT-4 and PaLM-2, have historically been categorized as "large," new developments in training techniques and data efficiency indicate that models like Phi-4 (14B) may be able to perform on par with much larger models, redefining the bar for LLM classification (Abdin et al., 2024; Hoffmann et al., 2022).

A Large Language Model is typically characterized by the following three key factors:

1. the number of parameters – The number of trainable weights in the model, historically considered the defining metric of largeness
2. the scale and quality of its training data – The breadth and curation of the dataset used for pretraining significantly impact generalization and performance.
3. its capacity for cross-task generalization - The ability of the model to function effectively on zero-shot, few-shot, and fine-tuning tasks with little modification

Historically, models with tens or hundreds of billions of parameters have been classified as LLMs, assuming that increased scale directly correlates with superior performance (Kaplan et al., 2020). However, recent research suggests that data quality and training methodology can enable smaller models to outperform larger counterparts (Hoffmann et al., 2022).

Phi-4, a 14-billion-parameter model, exemplifies this shift. While previous Phi models relied primarily on distillation from GPT-4, Phi-4 surpasses its teacher in reasoning-heavy benchmarks such as graduate-level problem-solving (GPQA) and mathematical reasoning (MATH) (Abdin et al., 2024). Its architecture remains similar to its predecessors, but improvements in synthetic data generation, post-training refinement, and training curriculum optimization allow it to compete with

much larger LLMs, including LLaMA-3-70B and GPT-4o-mini in select benchmarks (Microsoft Research, 2024).

Traditionally, the distinction between small and large language models has been parameter-centric, with models under 10 billion parameters considered "small" or "medium-sized" (Gunasekar et al., 2023). However, scaling laws indicate that data efficiency and architectural optimizations allow for "small" models by parameter count to function as large models in performance (Hoffmann et al., 2022).

Table 1
Performance of Phi-4 on a set of standard benchmarks.

Benchmarks	**Models**						
	Phi-4 14B	**Phi-3** 14B	**Qwen 2.5** 14B instruct	**GPT** 4o-mini	**LLaMA-3.3** 70B instruct	**Qwen 2.5** 72B instruct	**GPT** 4o
MMLU	84.8	77.9	79.9	81.8	86.3	85.3	**88.1**
GPQA	**56.1**	31.2	42.9	40.9	49.0	50.6	50.6
MATH	**80.4**	44.6	75.6	73.0	66.3	74.6	74.6
HumanEval	82.6	67.8	72.1	86.2	78.9	87.1	**90.6**
MGSM	80.6	63.9	77.9	86.5	89.1	82.8	**90.4**
SimpleQA	3.0	7.6	7.6	39.4	9.3	8.6	9.3
DROP	75.5	58.3	59.7	79.9	82.4	80.9	**85.6**
MMLUPro	70.4	51.3	63.2	63.4	69.6	69.6	**73.0**
HumanEval+	82.8	69.2	79.1	82.4	77.8	84.0	**88.0**
ArenaHard	75.4	67.0	68.3	73.1	76.4	79.2	**85.6**
LiveBench	47.6	28.1	49.8	58.7	57.1	64.6	**72.4**
IFEval	63.0	57.9	78.7	78.7	**89.3**	85.6	84.8
PhiBench (internal)	56.2	43.9	49.8	58.7	57.1	64.6	**72.4**

Data sourced from (Abdin et al., 2024). Best scores for each benchmark are highlighted in **bold**.

These results highlight that Phi-4 competes with or surpasses larger models, challenging the traditional parameter-based classification of LLMs. This performance is attributed to its advanced data curation, synthetic augmentation techniques, and novel post-training refinements, such as Direct Preference Optimization (DPO) and Pivotal Token Search (PTS), which improve its reasoning abilities and factual accuracy (Microsoft Research, 2024).

The emergence of efficiently trained models like Phi-4 calls for a reconsideration of what defines a large language model. While previous generations of LLMs emphasized parameter count as the primary metric, scaling laws suggest that training efficiency and data optimization can achieve comparable results at a fraction of the size (Hoffmann et al., 2022).

Phi-4's reliance on high-quality synthetic and curated datasets allows it to outperform models with much larger parameter counts, challenging the notion that size alone dictates performance. Post-Training techniques like Pivotal Token Search (PTS) enhance its ability to generate more accurate and reliable responses, narrowing the performance gap between Phi-4 and models exceeding 70B parameters (Microsoft Research, 2024). Unlike trillion-parameter models that require massive computational infrastructure, Phi-4 achieves LLM-level performance at a fraction of the cost, making it more accessible for real-world applications (Gunasekar et al., 2023).

In Conclusion traditional definitions of Large Language Models have been parameter-driven, models like Phi-4 challenge this paradigm by demonstrating LLM-level performance at a reduced scale. By leveraging data efficiency, architectural refinements, and synthetic data augmentation, Phi-4 competes with models 5× its size, suggesting that the classification of LLMs should now consider both parameter count and efficiency metrics. As AI research progresses, the emphasis will likely shift toward models that optimize performance while balancing computational feasibility, ensuring broader accessibility and responsible AI deployment.

Architectural and Functional Innovations of Phi-4

The Phi-4 model continues Microsoft's efforts to enhance natural language understanding and generation. Previous models like Turing-NLG and GPT-3 set benchmarks in terms of scale and capability, but Phi-4 aims to push these boundaries further by incorporating advanced learning techniques and broader datasets focusing on data quality (Abdin et al., 2024). Unlike its predecessors, which relied primarily on organic data, Phi-4 strategically incorporates synthetic data throughout the training process, leading to substantial improvements in performance, particularly in reasoning and problem solving tasks.

Phi-4 is a 14 billion parameter transformer-based LLM, developed with a specific focus on data quality. This substantial increase in parameters allows Phi-4 to capture and generate more nuanced and contextually accurate text. The model was trained using a mixed precision training approach, optimizing computational efficiency while maintaining high accuracy and stability. Advanced hardware optimizations, including the use of GPUs and TPUs, support large-scale computations. Innovations such as the Zero Redundancy Optimizer (ZeRO) and Megatron-LM further enhance the training process by effectively distributing the computational load.

The architecture of Phi-4 follows a decoder-only transformer with a default context length of 4096 tokens, extended to a 16K context length during midtraining. The training process involved approximately 10 trillion tokens, utilizing linear warm-up and decay schedules with a peak learning rate of 0.0003 and a global batch size of 5760. Additionally, the model employs techniques like rejection sampling and Direct Preference Optimization (DPO) during post-training, refining its outputs to achieve state-of-the-art performance.

Phi-4 demonstrates significant advancements in several key areas, which will be described in the following paragraph. One of the most important advantage is the models efficiency to performance ratio. Natural Language Understanding capabilities (Abdin et al., 2024). Additionally, Phi-4 generates coherent, context-sensitive text that closely mimics human writing, making it particularly useful for content creation, customer service automation, and interactive AI applications. This aspect of Phi-4's capabilities highlights its strength in Natural Language Generation (Abdin et al., 2024). The model's ability to learn from limited examples has been enhanced, enabling it to adapt quickly to new tasks with minimal additional training data, demonstrating its flexibility and efficiency in few-shot learning scenarios (Abdin et al., 2024). This can be very useful for our use case because of the limited size of the dataset. Furthermore, by building on its diverse training set, Phi-4 offers robust multilingual support, including less commonly spoken languages, thus broadening its applicability globally and enhancing its utility in a variety of linguistic contexts (Abdin et al., 2024). Leveraging its training on technical and programming data, Phi-4 can understand, generate, and even debug code snippets across various programming languages, highlighting its proficiency in code comprehension and generation (Abdin et al., 2024). Synthetic data plays a pivotal role in Phi-4's training regime. High-quality synthetic datasets are designed to prioritize reasoning and problem-solving and are meticulously generated using techniques like multi-agent prompting, self-revision workflows, and instruction reversal. By addressing some of the shortcomings of conventional unsupervised datasets, these techniques make it possible to create datasets that improve the model's capacity for reasoning and problem-solving.

The features of Phi-4 enable a wide range of applications in many industries. By producing comprehensive and context-specific automation scripts, Phi-4 can greatly simplify activities, lessen developer workload, and increase accuracy and consistency in SAP environments, where automating intricate and crucial processes is crucial.

In addition to being the result of recent advancements, Phi-4 serves as a basis for upcoming developments. Its efficiency, scalability, and capacity to adjust to domain-specific requirements are the goals of ongoing research. To further improve the model's resilience and protect data privacy, methods like federated learning and continuous training on decentralized data are being investigated. Microsoft's dedication to integrating cutting-edge AI capabilities throughout its product line is demonstrated by integration into platforms such as Visual Studio and Azure, which make these potent tools available to developers and businesses globally.

In conclusion, Microsoft's Phi-4 model pushes the limits of natural language creation and processing, marking a substantial breakthrough in AI technology. It serves as a foundation for upcoming AI-driven advancements due to its scalable architecture, broad range of capabilities, and varied application potential. The knowledge and technologies underpinning Phi-4 will be used as a benchmark as we further explore the solution architecture for this thesis, helping us create our refined, domain-specific models.

Implementation

meine Notizen zum inhalt des chapters Implementation: technology stack, code structure, integration, testing, deployment, dataset creation,

Dataset creation with ansible content parser

In this thesis, the initial phase of fine-tuning a LLM involved the development of a custom dataset designed to provide the model with the necessary training data. The primary emphasis was placed on Ansible code, particularly in the context of SAP environments. To ensure a sufficient volume of data for learning, general Ansible code was also incorporated to aid in syntax acquisition.

Ansible Lightspeed, developed by Red Hat, incorporates a robust open-source parser designed to efficiently transform data from a chosen GitHub repository into a JSONL (JSON Lines) file format. This file format is particularly advantageous for machine learning applications due to its structure. Each entry within the JSONL file is placed on a separate line, where each line comprises a JSON object. These objects are meticulously organized into multiple crucial columns, e.g. input and output. This separation ensures that the data is ready for training machine learning models, streamlining the process by clearly delineating the inputs and expected outputs, which are essential for supervised learning tasks. Furthermore, this structured approach aids in maintaining data integrity and enhancing the parser's operational efficiency when dealing with complex datasets from extensive repositories.

In order to generate the raw dataset, specific requirements need to be fullfiled defined by Red Hat. The parser does only support UNIX based operating systems, e.g. macOS or Linux. Additionally, Python 3.10 or higher must be installed on the system to ensure proper functionality. The first step is to install the most recent version of pip and ansible-content-parser library. The Python package manager pip is used to install and manage software packages written in Python. It is the standard tool for handling Python libraries and allows users to easily install dependencies from the Python Package Index (PyPI). In the context of this thesis, pip is used to install the ansible-content-parser library required for dataset generation. After installation, the parser can be executed via the terminal by providing the relevant repository URL and output directory. Hereby the REPO_URL refers to

the Git repository containing the Ansible content to be parsed, while OUTPUT_DIR specifies the directory in which the parsed data will be saved.

```
$ ansible-content-parser REPO_URL OUTPUT_DIR
```

The JSON element below represents a single entry example from the generated ftdata.jsonl file. Each row consists of several attributes, as mentioned before, with the most significant being the input and output fields. The input contains content extracted directly from the source repository, which is a YAML-formatted snippet such as an Ansible playbook section or task block. The output also originates from the same repository and contains the subsequent code segment that follows the input snippet. The dataset consists of multiple sections within the same playbook without duplicates.


```
{
"data_source_description": "",
"input": "---\n# Ansible Playbook for SAP BW/4HANA Sandbox installation\n\n# Use include_role / include_tasks inside Ansible Task block, instead of using roles declaration or Task block with import_roles.\n# This ensures Ansible Roles, and the tasks within, will be parsed in sequence instead of parsing at Playbook initialisation.\n\n\n#### Begin Infrastructure-as-Code provisioning ####\n\n- name: Ansible Play to gather input for gathering vars and VM provisioning\n  hosts: localhost\n  gather_facts: false\n\n  # pre_tasks used only for Interactive Prompts only and can be removed without impact\n  pre_tasks:\n\n    - name: Playbook Interactive - Check if standard execution with an Ansible Extravars file is requested by end user",
"license": "",
"module": "ansible.builtin.set_fact",
"output": "      ansible.builtin.set_fact:\n    playbook_enable_interactive_prompts: \"{{ true if (sap_vm_provision_iac_type is undefined and sap_vm_provision_iac_platform is undefined) else false }}\"\n",
```

```
"path":
 ↪  "deploy_scenarios/sap_bw4hana_sandbox/ansible_playbook.yml",
"repo_name": "ansible",
"repo_url":
 ↪  "https://github.com/sap-linuxlab/ansible.playbooks_for_sap"
}
```

The core elements of the dataset are the input and output columns, which serve as the primary components for the fine-tuning process of the language model. These columns are generated through the parsing process, where the Ansible Content Parser extracts code directly from the repositories and divides it into sections. Each section is then accordingly assigned to the input or output column, depending on the playbook structure,which essentially transforms the dataset into a sequence-to-sequence learning task, where the input column represents a partial or preceding code segment, and the output column contains the logically succeeding segment. This structure closely mimics real-world code generation and completion tasks, where a model is required to predict the next segment of code based on a given context. Consequently, the prompt provided to the model (the input) resembles the natural context that developers would provide when extending or completing an Ansible playbook, making this dataset particularly suitable for the targeted fine-tuning process. In order to ensure that the model also learns to handle complete playbook generation with questionnaire prompts, the dataset preparation process includes a step where the full Ansible playbook is reconstructed by concatenating the input and output columns. This concatenated form represents a complete, logically ordered playbook, providing the model with examples of fully assembled automation workflows. This dual approach — training on both partial segments and complete playbooks — aims to enhance the model's ability to generate syntactically correct and contextually appropriate Ansible code, even in cases where only partial context is available in the input.

Selection of relevant repositories

The selection of suitable repositories is an essential step when creating a high-quality dataset for training a LLM tailored to Ansible playbook generation. The quality, diversity, and relevance of the selected repositories directly influence the model's ability to generate meaningful and technically

correct code (Lozhkov et al., 2024). To ensure that only useful and representative data is included, a multi-stage selection process is applied. This process is inspired by the methodology used in the creation of The Stack v2 (Lozhkov et al., 2024), a dataset developed for training the StarCoder2 model.

The first step is to find a comprehensive source of open-source code repositories, namely those that provide Ansible playbooks for our particular use case (Lozhkov et al., 2024). This is crucial because the domain for which the model will be optimized must be reflected in the training data. Only the most recent version of each repository is taken into consideration to guarantee that only the most pertinent and recent code is used. Furthermore, only the main branch - which usually denotes the version of the representative codebase that is stable - is extracted.

A deduplication step comes after the repositories have been collected. Because many repositories on sites like GitHub are forks or almost identical copies of other projects, this is required. Overfitting, in which the model learns particular structures instead of broad patterns, might result from training on duplicate data. This is avoided by verifying the content of each repository and retaining only distinct repositories in the final dataset (Lozhkov et al., 2024).

The next stage in the dataset construction after deduplication is license verification. This is usually considered when ethical and legal concern involves in licensing of the training data, which is relevant when the training dataset consists of code repositories with potentially restrictive software licenses. In the context of this thesis, the focus is on academic fine-tuning and evaluation of the Phi-4 LLM, there is no legal requirement to comply as no code or model is distributed or commercially made available. As a result, regardless of their licensing state, all accessible repositories pertinent to the research topic are taken into account. The Starcoders license verification step excludes repositories with restrictive licenses like GPL or those without a clear license statement and only selects repositories with liberal licenses like MIT or Apache 2.0 to ensure compliance. For most of the repositories, where such information was missing, the authors used the ScanCode Toolkit to detect licenses at the file level, which were then classified according to permissiveness using identifiers and a custom propagation method (Lozhkov et al., 2024). The authors further filtered the dataset to include only permissively licensed and unlicensed files, explicitly excluding copyleft and commercial licenses due to associated legal risks (Lozhkov et al., 2024). Starcoders approach provides a framework for ensuring license compliance in open-source LLM publication. However,

the objectives of this thesis differ substantially from that context. In this work, the dataset was used exclusively for academic purposes, and no fine-tuned model, dataset, or code derived from the training data is distributed. The use of copyrighted or licensed material purely for internal research or noncommercial academic purposes is generally considered to fall under fair use or fair dealing. This interpretation is supported by current academic practices and aligns with the usage patterns allowed under many permissive open-source licenses, such as MIT, BSD, and Apache-2.0 Smith, 2022. Furthermore, as the model output was only analyzed in aggregate for evaluation purposes, and no training samples or generated completions are published, there is no redistribution of licensed content, and therefore no breach of licensing terms.

Afterwards, a manual inspection process is conducted. This involves reviewing samples from different repositories to visually assess their quality (Lozhkov et al., 2024). Manual inspection is particularly useful for identifying repositories that, while technically valid, contain code copied from tutorials or that exhibit poor coding practices unsuitable for training a high-quality model.

By following this structured, multi-stage selection process, the final dataset achieves a balance between relevance and quality. This approach ensures that the trained model learns from realistic, high-quality examples of Ansible playbooks, improving its ability to generate meaningful and practically useful code. This process is a technical necessity that reflects the increasing importance of responsible data sourcing in machine learning research.

Data processing and cleaning

duplicates in the dataset deleted, delete empty input / output rows, data split and ratio on train and eval etc.

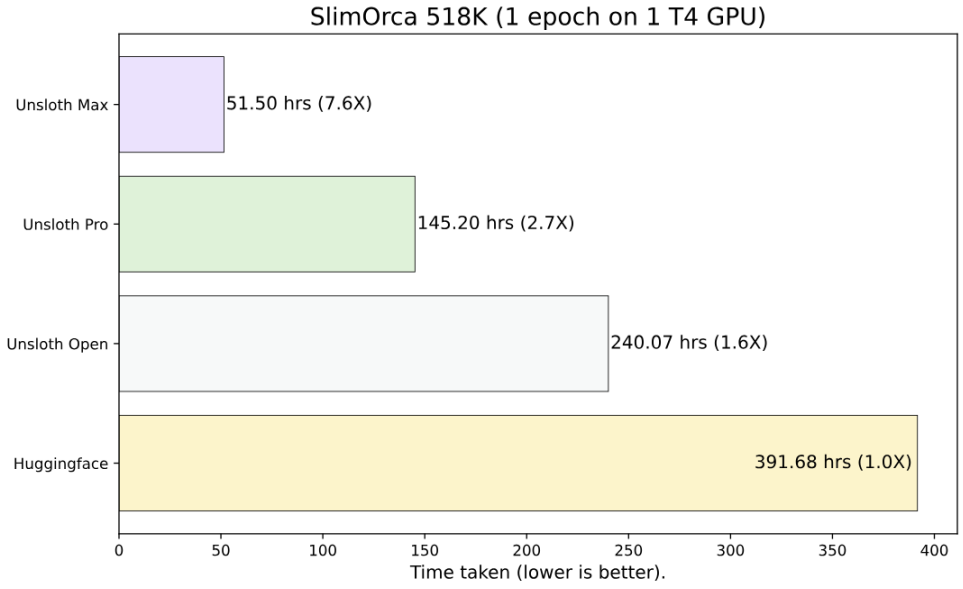
Technology stack and important libraries

In the following section, the core libraries and frameworks utilized in the implementation of this thesis are introduced. These components play a central role in enabling the fine-tuning of the Phi-4 large language model. Their selection is primarily driven by performance considerations, with a focus on maximizing computational efficiency, scalability, and compatibility with modern GPU architectures. Each library contributes to the overall training pipeline, either by optimizing memory usage, accelerating computations, or simplifying model integration.

Unsloth

The first and most important library utilized within the technology stack is Unsloth, which maximizes resource efficiency. Unsloth claims to reduce memory usage by 60 percent, while enabling six times larger batch sizes. This is achieved through software optimizations that utilize the given resource capacities. Optimizing autograd for efficient LoRA fine-tuning is crucial. When computing the gradients of a function coupled with Low-Rank Adaptation (LoRA) adapters, six matrix differentials must be computed due to the freezing of the original weight matrices. In PyTorch, these gradients are automatically derived by saving all operations during forward propagation as a computational graph, which is then used for backpropagation. However, this process increases computation time and memory usage due to the storage of the computational graph, reducing overall efficiency. With Unsloth's manual autograd, gradients are derived manually, eliminating the need to store and use a computational graph. This reduces memory usage and increases performance, as no automatic gradient computation is required. The downside of manual derivation is that Unsloth has to implement these for each LLM, hence Unsloth has limited selection of LLM's. Figure 4 shows the performance in training time on identical dataset and hardware by comparing the same HuggingFace model and the Unsloth optimized model.

Figure 2
Training time comparision



Unsloth Max achieves up to 7.6x speedup over Hugging Face. Image adapted from Unsloth (AI, 2024).

PyTorch Framework

PyTorch is an open-source machine learning library developed by Facebook's AI Research lab (FAIR). It provides two high-level features that are critical for modern machine learning workflows: a tensor computation library with strong GPU acceleration support and an automatic differentiation engine for building and training neural networks (Paszke et al., 2019). Unlike traditional machine learning libraries, PyTorch supports dynamic computation graphs, which allow developers to define, modify, and execute the network structure at runtime. This approach facilitates a high level of flexibility, making PyTorch particularly suited for research and prototyping tasks. Additionally, the PyTorch ecosystem provides three domain-specific libraries that extend its functionality for specialized machine learning tasks. These include torchvision for computer vision applications, torchaudio for audio signal processing and feature extraction, and torchtext for natural language processing (NLP) workflows. In the context of this thesis, the torch library is utilized, initializing a tensor to store predictions, ensuring it has the correct shape to accommodate the batch size and the number of classes the model predicts. This initialization is done for the evaluation step of the LLM where you might encounter cases where the model doesn't produce any predictions for certain examples. In such cases, having a zero-filled tensor ensures a placeholder for these empty predictions, which allows to proceed with metrics calculations without encountering errors due to missing values.

```
# Ensure logits are always tensors, even if generation is empty
if isinstance(predictions,
 ↪  FastLanguageModel.models._utils.EmptyLogits):
    predictions = torch.zeros((labels.shape[0],
     ↪  model.config.num_labels), dtype=torch.float32)
    predictions = predictions.to(labels.device)
```

CUDA GPU Acceleration

CUDA (Compute Unified Device Architecture) is a parallel computing platform and programming model developed by NVIDIA. It enables developers to harness the full power of NVIDIA GPUs for general-purpose computing. CUDA allows for massive parallelization of mathematical opera-

tions that are essential for deep learning, such as matrix multiplications, convolutions, and tensor transformations.

In the context of PyTorch, CUDA is used to accelerate the forward and backward passes during model training. When enabled, PyTorch automatically offloads eligible operations to the GPU, significantly reducing training time, especially for large models and datasets. In order to ensure compatibility between PyTorch and CUDA, precompiled binaries are made available for different CUDA versions, which is important since not all PyTorch versions are compatible. For example, in this thesis, the following installation command is used to install PyTorch with CUDA 12.4:

```
$ pip install torch torchvision torchaudio torchtext --index-url
↪  https://download.pytorch.org/whl/cu124
```

This command retrieves optimized PyTorch binaries from the official PyTorch repository that are compatible with CUDA 12.4. The use of this custom index URL ensures that the installed packages can leverage GPU acceleration effectively, provided that the system includes a compatible NVIDIA GPU and driver.

Evaluate and NumPy: Calculation libraries

The evaluate library is an open-source Python package developed by Hugging Face, designed to simplify the computation of evaluation metrics in machine learning workflows. It provides a standardized interface for accessing a wide range of commonly used metrics across various domains, such as natural language processing, computer vision, and structured prediction tasks. A key advantage of the evaluate library is its integration with the Hugging Face ecosystem and its compatibility with popular datasets and models. It enables users to load metrics with a single command and apply them to predictions and references with minimal code.

```
1 # Install and load ROUGE score
2 pip install rouge_score
3 from evaluate import load
4 rouge = load("rouge")
```

```
# Calculate ROUGE score
rouge_results = rouge.compute(predictions=modified_predictions,
    references=decoded_references, use_aggregator=True)
```

Internally, many of the metrics are implemented using highly optimized libraries such as scikit-learn or sacrebleu, which ensures both reliability and efficiency. In the context of this thesis, evaluate is utilized to measure the performance of the fine-tuned Phi-4 model. Specific metrics such as ROUGE, BLEU, METEOR, and CHRF are employed to assess the quality of generated Ansible code sequences in comparison to reference completions. These metrics and their results are discussed in detail in the Evaluation chapter 5. In addition to these standard measures, a custom evaluation metric based on ansible-lint is also integrated to assess syntactic correctness from a domain-specific perspective.

Numerical Python - NumPy - is a fundamental package for scientific computing in Python. It provides a high-performance, multidimensional array object called the ndarray, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy is widely regarded as the backbone of the Python data science ecosystem and is used extensively in fields such as machine learning, data analysis, and numerical simulations. One of NumPy's core advantages is its ability to perform element-wise operations and broadcasting over large arrays without the need for explicit loops, which significantly improves performance compared to standard Python lists. Furthermore, NumPy offers tools for integrating C/C++ and Fortran code, which allows for low-level optimization and compatibility with other scientific computing libraries. In the context of this thesis, NumPy is primarily used for preprocessing and transforming dataset elements, statistical evaluations, and performing lightweight mathematical operations prior to model input or during result analysis. Its efficient memory model and vectorized computation capabilities make it an ideal choice for handling large-scale data during the model fine-tuning process.

Hardware Resources

Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet. Lorem ipsum dolor sit amet, consetetur sadipscing elitr, sed diam nonumy eirmod tempor

invidunt ut labore et dolore magna aliquyam erat, sed diam voluptua. At vero eos et accusam et justo duo dolores et ea rebum. Stet clita kasd gubergren, no sea takimata sanctus est Lorem ipsum dolor sit amet.

Fine-Tuning Process

Model Initialization and Configuration

Integration of Unsloth and LoRA

Training Procedure and Hyperparameters

Evaluation Integration

Checkpointing and Model Saving

Evaluation

eval metrics and qualitative survey with experts on the code

Discussion

endresults zukunftsperspective

Conclusion

Bibliography

Abdin, M., Aneja, J., Behl, H., Bubeck, S., Eldan, R., Gunasekar, S., Harrison, M., Hewett, R. J., Javaheripi, M., Kauffmann, P., Lee, J. R., Lee, Y. T., Li, Y., Liu, W., Mendes, C. C. T., Nguyen, A., Price, E., de Rosa, G., Saarikivi, O., ... Zhang, Y. (2024). Phi-4 technical report. https://arxiv.org/abs/2412.08905

AI, U. (2024). Introducing unsloth: Faster and more efficient llm fine-tuning [Accessed: 16-Mar-2025]. https://unsloth.ai/introducing

Amazon. (2023). Ai powered productivity tool [Accessed: January 17, 2025]. https://aws.amazon.com/codewhisperer/

Ansible. (2024). Ansible documentation [Accessed: June 25, 2024]. https://docs.ansible.com/ansible/latest/index.html

Bass, L., Weber, I., & Zhu, L. (2015). *Devops: A software architect's perspective*. Addison-Wesley Professional.

Geerling, J. (2015). *Ansible for devops: Server and configuration management for humans*. Leanpub.

GitHub. (2021). Github copilot [Accessed: January 17, 2025]. https://github.com/features/copilot

Gupta, S., Qian, X., Bhushan, B., & Luo, Z. (2019). Role of cloud erp and big data on firm performance: A dynamic capability view theory perspective. *Management Decision*, *57*(8), 1857–1882.

Hevner, A., Chatterjee, S., Hevner, A., & Chatterjee, S. (2010). Design science research in information systems. *Design research in information systems: theory and practice*, 9–22.

Howard, J., & Ruder, S. (2018). Universal language model fine-tuning for text classification.

Klaus, H., Rosemann, M., & Gable, G. G. (2000). What is erp? *Information systems frontiers*, *2*, 141–162.

Lozhkov, A., Li, R., Allal, L. B., Cassano, F., Lamy-Poirier, J., Tazi, N., Tang, A., Pykhtar, D., Liu, J., Wei, Y., et al. (2024). Starcoder 2 and the stack v2: The next generation. *arXiv preprint arXiv:2402.19173*.

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of management information systems*, *24*(3), 45–77.

Poston, R., & Grabski, S. (2000). The impact of enterprise resource planning systems on firm performance.

Pujar, S., Buratti, L., Guo, X., Dupuis, N., Lewis, B., Suneja, S., Sood, A., Nalawade, G., Jones, M., Morari, A., & Puri, R. (2023). Automated code generation for information technology tasks in yaml through large language models. https://arxiv.org/abs/2305.02783

Replit. (2016). Build software collaboratively with the power of ai [Accessed: January 17, 2024]. https://replit.com/

Sahoo, P., Pujar, S., Nalawade, G., Genhardt, R., Mandel, L., & Buratti, L. (2024). Ansible lightspeed: A code generation service for it automation. *Proceedings of the 39th IEEE/ACM International Conference on Automated Software Engineering*, 2148–2158. https://doi.org/10.1145/3691620.3695277

Smith, P. M. (2022, October). 71copyright, contract, and licensing in open source. In *Open source law, policy and practice*. Oxford University Press. https://doi.org/10.1093/oso/9780198862345.003.0003

Tab9. (2018). Tab9 ai coding assistant [Accessed: January 17, 2025]. https://www.tabnine.com/

UCC, S. (2024). Sap ucc documentation [Accessed: October 10, 2024]. https://ucc.tum.de/

Ziegler, A., Kalliamvakou, E., Simister, S., Sittampalam, G., Li, A., Rice, A., Rifkin, D., & Aftandilian, E. (2022). Productivity assessment of neural code completion. https://arxiv.org/abs/2205.06537

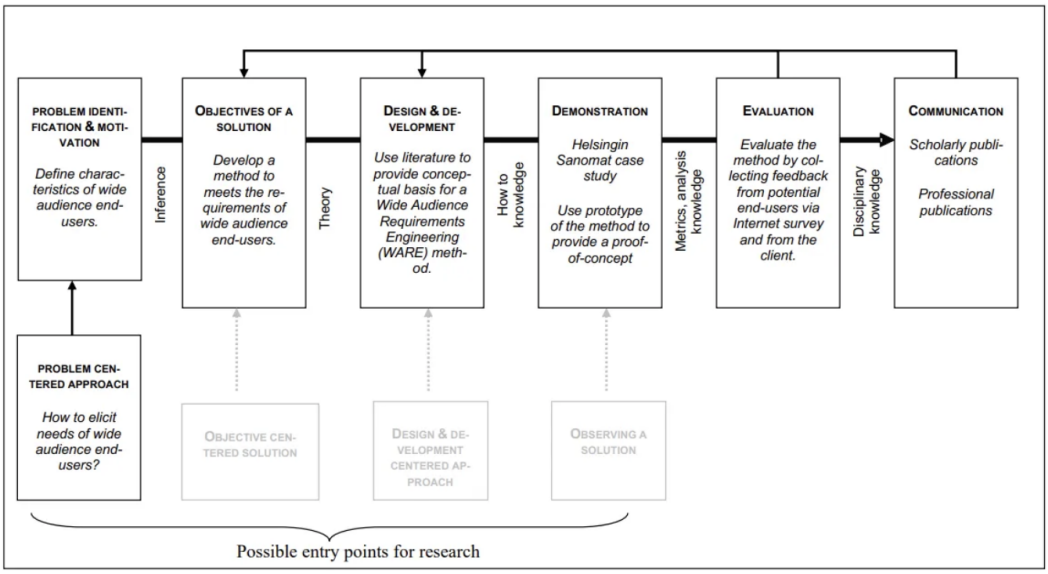
Appendix

Table 2
Performance of Phi-4 on a set of standard benchmarks.

Benchmarks	**Models**						
	Phi-4 14B	**Phi-3** 14B	**Qwen 2.5** 14B instruct	**GPT** 4o-mini	**LLaMA-3.3** 70B instruct	**Qwen 2.5** 72B instruct	**GPT** 4o
MMLU	84.8	77.9	79.9	81.8	86.3	85.3	**88.1**
GPQA	**56.1**	31.2	42.9	40.9	49.0	50.6	50.6
MATH	**80.4**	44.6	75.6	73.0	66.3	74.6	74.6
HumanEval	82.6	67.8	72.1	86.2	78.9	87.1	**90.6**
MGSM	80.6	63.9	77.9	86.5	89.1	82.8	**90.4**
SimpleQA	3.0	7.6	7.6	39.4	9.3	8.6	9.3
DROP	75.5	58.3	59.7	79.9	82.4	80.9	**85.6**
MMLUPro	70.4	51.3	63.2	63.4	69.6	69.6	**73.0**
HumanEval+	82.8	69.2	79.1	82.4	77.8	84.0	**88.0**
ArenaHard	75.4	67.0	68.3	73.1	76.4	79.2	**85.6**
LiveBench	47.6	28.1	49.8	58.7	57.1	64.6	**72.4**
IFEval	63.0	57.9	78.7	78.7	**89.3**	85.6	84.8
PhiBench (internal)	56.2	43.9	49.8	58.7	57.1	64.6	**72.4**

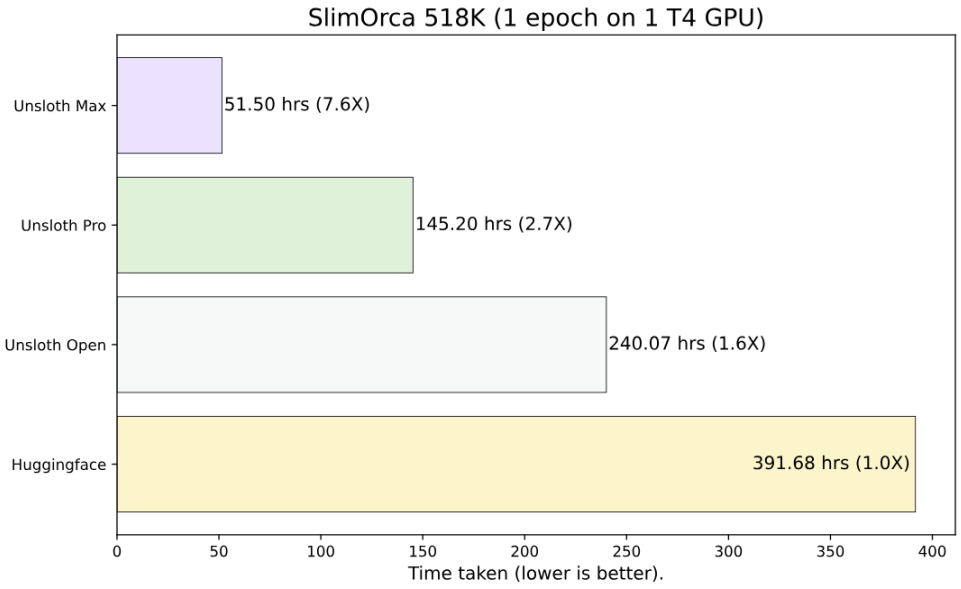
Data sourced from (Abdin et al., 2024). Best scores for each benchmark are highlighted in **bold**.

Figure 3
DSR process cycles



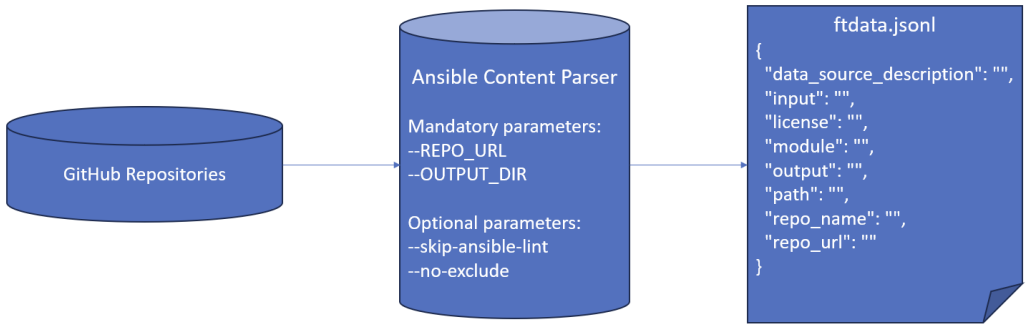
Overview of the iterative phases in DSR process (Peffers et al., 2007).

Figure 4
Training time comparision



Unsloth Max achieves up to 7.6x speedup over Hugging Face. Image adapted from Unsloth (AI, 2024).

Figure 5
ansible content parser process



GitHub repository for Ansible Content Parser with input and output configuration details