Chapter 1 - Introduction

1.1 Background

The modern technological world is driven by the software development industry, and millions of software engineers worldwide are among the highest-paid professionals. Companies invest heavily in this workforce to develop and maintain software, leading to substantial labor costs.

Recently, large language models (LLMs) have emerged as powerful tools capable of automating code generation from natural language descriptions, thus enhancing the software engineering efficiency.

According to (McKinsey Report, 2023), generative AI can significantly decrease the time developers spend on coding - by up to 45% - which can enable companies to reduce labor costs. This efficiency gain is particularly impactful for large-scale projects where even marginal improvements translate into significant cost reductions.

Automating code generation also leads to improved code quality through consistent adherence to coding standards and best practices. Studies like (Almeida Y. et al, 2024) and (Martinović B. and Rozić R., 2024) highlight how consistent and well-structured code generated by AI-enhanced tools contributes to minimizing human errors and bugs. In (Kalliamvakou, 2024), GitHub concluded that developers using GitHub Copilot finished their tasks 55% faster than the developers who preferred not to use GitHub Copilot. But developer productivity goes beyond speed - according to the same study between 60–75% of developers reported that they feel more fulfilled with their job, feel less frustrated, and can focus on more satisfying work when using GitHub Copilot.

In accordance with (Gartner Report, 2023), accelerated development cycles allow companies to bring products to market more quickly, providing a significant competitive edge by reducing manual coding time and streamlining development processes.

According to Google's CEO Sundar Pichai (Pichai, 2024) AI tools are already having a sizable impact on software development, and more than 25% of new code at Google is AI-generated. This helps Google engineers achieve more and work faster.

Google developers aren't the only programmers using AI to assist with coding tasks. According to Stack Overflow's 2024 Developer Survey (Stack Overflow, 2024), over 76% of all respondents are already using or plan to use AI tools in the development process this year, with 62% actively using them. A 2023 GitHub survey (Shani S. & GitHub Staff. 2023) showed that 92 percent of US software engineers are using AI tools for coding tasks in and outside of work.

However, the use of proprietary LLMs poses significant challenges regarding sensitive data protection and intellectual property rights. Developers often need to use proprietary or confidential information either to train these models or at the time of inference, risking data breaches and unauthorized access to intellectual property. This not only jeopardizes a company's competitive advantage but also exposes it to legal liabilities.

To address these concerns, companies could use small language models (SLMs) boosted by agents and deployed in resource-constrained and secure environments. SLM-based agents offer a cost-effective and privacy-preserving alternative to proprietary LLMs. They enable organizations to automate the creation of basic code routines without compromising sensitive data, which reduces the time developers spend on manual coding.

This approach is especially beneficial for understaffed projects, providing efficient solutions without the need to hire additional software engineers for routine tasks.

Leveraging SLM-based agents for automated code generation addresses the dual challenge of making the process of writing code more efficient and protecting sensitive data. Furthermore, companies with limited financial resources that prefer not to hire many software engineers, can leverage this technology to efficiently write code while using a small workforce and achieve what would have been impossible just several years ago. This approach enables companies to optimize developer productivity, enhance code quality, and accelerate time-to-market, all while ensuring data confidentiality.

1.2 Research Motivation

The primary motivation for this research is ensuring efficiency, cost reduction, and data privacy in software development. As someone who worked for several large companies that had classified proprietary information or intellectual property, we can state that there is an evident trend that companies are reluctant to use LLMs for data privacy and security reasons.

While LLMs have demonstrated remarkable capabilities in automating code generation from natural language descriptions, they pose significant challenges related to sensitive data protection and intellectual property rights. Using proprietary LLMs often requires transmitting confidential information to third-party servers, raising concerns about data breaches and unauthorized access to proprietary code. This not only risks a company's competitive advantage but also exposes it to potential legal liabilities.

On the other hand, these companies could use SLMs to provide a similar level of solution quality. By conducting this research, we aim to develop a solution that leverages the advantages of SLMs while minimizing their limitations compared to LLMs. To address these challenges, there is a strong motivation to explore the use of SLMs enhanced by agents for automated code generation within secure, resource-constrained environments. SLM-based agents offer several compelling benefits:

1. **Deploying SLMs in-house** ensures that sensitive data and intellectual property remain within the organization's secure environment.
2. **SLMs are generally more cost-effective** than proprietary LLMs which makes advanced code generation capabilities more accessible to organizations with limited resources.
3. SLMs don’t require **massive GPU clusters** to fine-tune.
4. SLM-based agents can consistently adhere to **coding standards and best practices**, reducing human errors and bugs.
5. **Faster development cycles** enable companies to bring products to market more quickly, providing a competitive edge in rapidly evolving industries.
6. In understaffed projects, SLM-based agents can compensate for **limited human resources** by efficiently generating code, reducing the need to hire additional developers for routine tasks.
7. Researching how to enhance SLMs with agent-based architectures **contributes to the broader field of AI and machine learning**, pushing the boundaries of what smaller models can achieve in specialized tasks like code generation.

1.3 Problem Statement

Using proprietary large language models (LLMs) to automatically generate code is costly and not safe from the sensitive data protection and intellectual property standpoints forcing developers to spend twice as much time writing code manually.

Proprietary LLMs are expensive in deployment and/or inference and expose sensitive data, pushing teams to code manually, slowing development and increasing costs. Data privacy and intellectual property risks with proprietary LLMs discourage their use, compelling developers to spend more time coding manually

1.4 Thesis Statement

Agents based on open-source small language models (SLM) deployed in resource-constrained environments for automated code generation will ensure lower costs and sensitive data protection, reducing the manual coding time and speeding up development cycle.

By paving the road for automated code generation, SLM-based agents reduce the overall time developers spend writing code while still preserving data privacy. This research introduces a novel approach by leveraging SLM-based agents to automate code generation from natural language descriptions, surpassing SLMs and attempting to approach the proprietary LLMs in code quality. Python software developers may use such a product to automatically generate code while ensuring sensitive data protection and reducing time for manual coding.

1.5 Research Objectives

The primary objective of this research is to develop and evaluate an agent-based system utilizing SLMs to automatically generate code from natural language descriptions. The study aims to bridge the performance gap between SLMs and proprietary LLMs in code generation tasks while ensuring data privacy and cost efficiency.

Since LLMs are costly and require investments into training data and large GPU clusters, companies can deploy more accessible SLMs. A tradeoff would be the lack of quality of LLMs, but using agents can make it competitive with LLMs to a certain degree Hence, the study aims to enhance and evaluate the code generation quality while ensuring data privacy and security, improving the cost efficiency and developer productivity, and accelerating time-to-market.

By achieving these objectives, the research aims to create a viable, secure, and efficient alternative to proprietary LLMs for automated code generation.

1.6 Research Questions and Hypotheses

Below is a list of research questions studied in the current Praxis, as well as hypotheses that need to be proved in the end of the Praxis cycle.

Research question 1: Will fine-tuning SLMs used by agents result in higher code generation quality as measured by the maintainability index?

Research question 2: Will changing SLM parameters, such as temperature and top-p, ensure greater code quality based on lower cyclomatic complexity??

Research question 3: Which agentic workflow, reflection or multi-agent collaboration, leads to a greater number of tests passed?

Hypothesis 1: Fine-tuning SLMs on domain-specific data will noticeably increase the maintainability index compared to using an LLM without fine-tuning.

Hypothesis 2: Adjusting SLM parameters, such as temperature and top-p, will noticeably improve the cyclomatic complexity of auto-generated code.

Hypothesis 3: Multi-agent collaboration will lead to a noticeably greater number of tests passed compared to the reflection agentic workflow.

1.7 Scope of Research

This research aims to assess the feasibility and competitiveness of using SLMs enhanced with agent-based architectures for automated code generation from natural language descriptions. It involves the following key activities:

- **Establishing the current benchmarks for code generation** by LLMs and SLMs using public leaderboards.
- **Selecting one or several SLMs** which can be used as is or which can be additionally fine-tuned on public code generation datasets in order to enhance their code generation capabilities.
- **Developing agent-based architectures** that integrate with SLMs to enhance their reasoning, planning, and problem-solving abilities facilitating the decomposition of complex coding tasks into manageable subtasks, enabling iterative refinement, and incorporating feedback mechanisms to improve code generation outputs.
- **Conducting systematic experiments** to assess the performance of the enhanced SLMs in automated code generation tasks on a variety of coding challenges based on natural language descriptions.

- **Utilizing quantifiable evaluation metrics** to evaluate the quality, correctness, and efficiency of the generated code.
- **Documenting the methodologies, experiments, and findings** comprehensively to contribute to the academic community.

1.8 Research Limitations

This research on SLMs enhanced by agent-based architectures for automated code generation has several limitations that may impact the scope, applicability, and generalizability of the findings. Recognizing these limitations is essential for interpreting the results accurately and identifying areas that require further investigation.

First of all, a fundamental limitation of this study is the fact that due to their smaller size and fewer training parameters, SLMs may not achieve the same level of sophistication, contextual understanding, and code generation quality as LLMs. Despite enhancing SLMs with agentic workflows, there may still be a noticeable gap in complex code generation tasks where LLMs excel. Also, the research focuses exclusively on SLMs and does not include the implementation of similar experiments using LLMs. Any comparisons drawn between SLM-based agents and proprietary LLMs rely on existing literature or reported benchmarks.

The study is conducted within the confines of limited hardware and computational resources, which are representative of resource-constrained environments typical for organizations without extensive infrastructure. This restricts the extent of model fine-tuning, the size of datasets processed, and the complexity of agentic architectures employed which could impact the final results. The one year allocated for this research may limit the depth and breadth of exploration possible - not all SLMs, programming languages, agentic workflows, or evaluation metrics can be exhaustively examined during this relative short period of time. That is why this study is confined to Python code generation and specific agentic workflows — namely, reflection and multi-agent collaboration — which may not capture the full spectrum of potential strategies. Also, the research specifically focuses on code generation from natural language descriptions and does not address other aspects of software engineering automation, such as code refactoring, bug detection, or code optimization.

The datasets used for fine-tuning and evaluating SLMs are limited to publicly available ones. The absence of proprietary, domain-specific, or larger-scale datasets may affect the models' ability to generalize to real-world applications, and the quality and diversity of these datasets may influence performance outcomes. In addition, SLMs trained on publicly available data may inadvertently learn and propagate biases present in the training data. The research does not specifically address bias detection or mitigation strategies, which could impact the fairness, ethical considerations, and acceptance of the generated code in sensitive applications.

Although a key motivation for using SLMs is to enhance data privacy by keeping computations in-house, the research does not delve deeply into the implementation of robust data protection measures. Additionally, the rapid evolution of AI technologies means that newer SLMs or alternative methods may emerge by the time of publication, potentially surpassing the models and approaches evaluated in this study and affecting the relevance and applicability of the findings.

1.9 Organization of Praxis

This Praxis has the following structure: the current *Introduction* chapter will be followed by Chapter 2 *Literature Review* describing the research performed in the field to date to solve similar problems. It includes a careful, but critical comparison of available work described in the literature that is directly related to the problem at hand.

Chapter 3 *Research Methodology* conveys a complete understanding of the methodology used to conduct the research capturing assumptions, ease of use, input data, expected output results, constraints, required adaptations, and other important aspects.

Chapter 4 *Results* demonstrate the actual outputs of the steps described in the methodology highlighting the results accomplished after each step of the methodology. It may contain descriptive statistics, charts, tables, and other visual representations of the work conducted in the Praxis. This chapter also summarizes key findings and compares results of various methods examined, final performance, etc.

Chapter 5 *Discussion and Conclusions* outlines how the findings of the study are related to the research questions and hypotheses.

The *References* section lists all information sources used to justify or conduct the research or which were mentioned / cited in the Praxis.

References

McKinsey Report. 2023. Unleashing developer productivity with generative AI.

Yonatha Almeida, Danyllo Albuquerque, Emanuel Dantas Filhob, Felipe Munizb, Katyusco de Farias Santosb, Mirko Perkusichc, Hyggo Almeidac, Angelo Perkusichc. AICodeReview: Advancing code quality with AI-enhanced reviews. 2024.

Boris Martinović and Robert Rozić. Impact of AI Tools on Software Development

Code Quality. 2024.

Eirini Kalliamvakou, GitHub. 2024. Quantifying GitHub Copilot’s impact on developer productivity and happiness.

Shani S. & GitHub Staff. 2023. Survey reveals AI’s impact on the developer experience.

Gartner Report. 2023. Top Strategic Technology Trends. http://emtemp.gcom.cloud/ngw/globalassets/en/publications/documents/2023-gartner-top-strategic-technology-trends-ebook.pdf

Pichai, S. 2024. Q3 earnings call: CEO’s remarks. https://blog.google/inside-google/message-ceo/alphabet-earnings-q3-2024/#full-stack-approach

Morris, S. 2023. AI, cloud boost Alphabet profits by 34 percent. https://arstechnica.com/gadgets/2024/10/ai-cloud-boost-alphabet-profits-by-34-percent/

Stack Overflow. 2024. AI. https://survey.stackoverflow.co/2024/ai