[Image: University Logo - IIUC]

International Islamic University Chittagong

Department of Computer Science and Engineering

A Thesis Proposal on

Building Predic: A Decentralized, Language-Specific Code Completion System

Submitted By:

Name	Student ID
Mir Md. Tarhimul Quader	C221017
Md. Iftaker Ahamed Sayem	C221020
Turja Dutta	C221026

1 Introduction

Intelligent tools like code completion systems are highly beneficial for modern software development, increasing productivity and reducing cognitive load. While traditional local code completion tools (e.g., based on Abstract Syntax Trees or keyword matching) offer speed, they often lack deep contextual understanding and cannot suggest complex or novel code structures effectively. Conversely, many advanced systems, particularly those based on large language models (LLMs), are often too resource-intensive for environments with limited computational power or raise privacy concerns due to centralized data processing.

This thesis proposes "Predic," a **lightweight (aiming for model sizes under 500MB and inference latencies below 100ms on typical developer hardware)** and decentralized code completion system. Predic aims to provide effective, language-specific code suggestions by utilizing Small Language Models (SLMs) tailored for common programming languages. An initial base model will be trained on publicly

available code by the research team. Subsequently, to enable personalization, continuous improvement, and ensure user data privacy, Predic will leverage **Federated Learning (FL)** for fine-tuning, where individual users (acting as clients) can opt-in to refine models using their local, private code. The system's practicality will be demonstrated through a Visual Studio Code extension designed for real-time, local code completion.

2 Research Background

2.1 Problem Statement

While traditional local code completion tools (e.g., based on Abstract Syntax Trees or keyword matching) offer speed, they often lack contextual understanding and cannot suggest complex or novel code structures effectively. Conversely, large language models (LLMs) have shown success in code completion, but their generalized nature and high computational demands make them impractical for many real-world development environments, especially those requiring specialized knowledge of particular languages or operating with limited resources. Furthermore, centralized training data requirements can pose risks when dealing with private or sensitive code.

There is a clear need for a lightweight, effective, and privacy-preserving code completion solution that can provide accurate, language-specific recommendations without the overhead of monolithic LLMs. This requires an approach that balances model efficiency, specialization, and data security through mechanisms like federated fine-tuning on user-specific data.

3 Literature Review

This research builds upon the foundational concepts and advancements presented in the following works:

- **Full Line Code Completion** (Anton et al., 2024): Bringing AI to Desktop: Highlights the need for fast, reliable, and private code completion that works fully on a user's computer, as many AI-driven tools are online or lack privacy. This aligns with Predic's goal of local execution.
- **Context Composing for Full Line Code Completion** (Anton Semenkin, 2024): Notes that developers often prefer accurate single-line completions, especially when run locally, guiding our focus for efficient local suggestions provided by SLMs.
- **Attention Is All You Need** (Vaswani et al., 2017): Introduced the Transformer architecture, whose self-attention mechanism is a core component for modern language models, including the SLMs to be developed for Predic.
- **Language Models are Unsupervised Multitask Learners** (Radford et al., 2019): Demonstrated the power of transformer-based language models to perform various tasks without explicit supervision, forming a basis for pre-training effective code models for our initial central training phase.

- **Flower: A Friendly Federated Learning Research Framework** (Beutel et al., 2020): Provides a versatile framework for implementing Federated Learning. We will leverage Flower to manage the decentralized fine-tuning of our SLMs on users' private code, thereby enhancing privacy and personalization.
- **ESPnet-ONNX: Bridging a Gap Between Research and Production** (Someki et al., 2022): Showcases methods for converting and optimizing research models (e.g., to ONNX format) for efficient deployment, relevant for packaging our trained SLMs into a responsive VS Code extension.
- **Code Llama: Open Foundation Models for Code** (Rozière et al., 2023): Presents highly capable open-source foundation models for code, offering insights into training methodologies, data curation, and evaluation techniques that we will adapt for developing our specialized, smaller SLMs.

4 Methodology

4.1 Model Design

The core of Predic will consist of one or more Small Language Models (SLMs), each based on the Transformer architecture. For language-specificity, we plan to either:

1. Train separate, compact SLMs for each target programming language (e.g., a Python SLM, a JavaScript SLM).
2. Train a base SLM on a mixed-language dataset and then fine-tune it for specific languages.

The design will prioritize computational efficiency for local execution and responsiveness, aiming for model sizes under 50MB and inference latencies below 100ms.

4.2 Architecture

The proposed system architecture will comprise:

1. **Language-Specific SLMs:** Compact Transformer-based models, each optimized for code completion in a specific programming language. These models will include standard components:
    - **Tokenizer:** Adapted for the syntax of the specific programming language.
    - **Embedding Layer.**
    - **Transformer Encoder/Decoder Blocks.**
    - **Output Layer:** For predicting code token sequences.
2. **Federated Learning Framework:** Utilizing **Flower** to manage the decentralized fine-tuning of the SLMs. In this phase, individual users (clients) with the VS Code extension can opt-in to contribute updates from their local, private code, improving the model without sharing their raw data. The initial base model training will occur centrally using public datasets.

3. **VS Code Extension:** The client-side interface that hosts the SLM(s) locally, interacts with the editor to get context, and provides completion suggestions.

4.3 Implementation

The SLMs will be implemented and trained using frameworks like PyTorch or TensorFlow.

- **Training:** A hybrid approach will be employed:
    i. **Base Model Training:** Initial SLMs will be trained centrally by the research team using publicly accessible code from GitHub repositories.
    ii. **Federated Fine-Tuning:** Once deployed via the VS Code extension, users can opt-in to participate in Federated Learning using the Flower framework. These users (clients) will contribute to further fine-tuning the SLMs on their local, private code relevant to the programming languages they use, enhancing model personalization and performance.
- **Model Optimization:** Techniques will be explored to ensure the SLMs are lightweight and fast for inference, potentially using model quantization or pruning, and conversion to formats like ONNX for efficient deployment.
- **Language Support:** Initially, SLMs for a few popular programming languages (e.g., Python, JavaScript) will be developed.

4.4 Deployment

Predic will be deployed as a Visual Studio Code (VSCode) extension. The trained SLM(s) will run locally on the user's machine, ensuring privacy and low-latency suggestions. This local deployment facilitates real-world evaluation and usage.

4.5 Research Type

Applied and Quantitative Research: This study focuses on the practical application of developing an efficient code completion tool. Quantitative methods will be used to evaluate its performance, including:

- **Accuracy:** Metrics like exact match and pass@k (e.g., pass@1, pass@3) for code suggestions.
- **Latency:** Inference time for generating suggestions (target < 100ms).
- **Efficiency:** Model size (target < 500MB) and memory footprint.

Performance will be benchmarked through systematic experimentation, potentially comparing against existing non-AI local completion and a non-federated version of our SLM.

4.6 Data Sources

Initial base model training data will consist of publicly accessible code from GitHub repositories for the selected programming languages (e.g., Python, JavaScript). Preprocessing will focus on extracting

relevant code snippets suitable for training code completion models, such as single lines or function bodies. The diversity of coding styles in these repositories will help in training robust base SLMs. **For the Federated Learning phase,** training data will be sourced (with explicit user opt-in and privacy-preserving mechanisms) from the local development environments of participating users. This data, consisting of their private code, will be used to fine-tune the models locally on the user's machine, with only model updates being shared via the Flower framework, not the raw code itself.

5 Conclusion

This thesis proposes Predic, a system designed to provide lightweight, decentralized, and language-specific code completion. By utilizing Small Language Models, initially trained on public datasets and then fine-tuned via Federated Learning on opted-in user data, Predic aims to address the limitations of resource-intensive, centralized LLMs and the contextual shortcomings of traditional non-AI completion tools. It offers a practical and privacy-preserving solution for developers. The implementation as a Visual Studio Code extension will demonstrate its utility and potential contribution to intelligent software development tools, focusing on efficient, local code assistance.