

1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects - DevOps engineers - Developers

Problem, context, and related work Software engineering today is a challenging task. As the complexity of software keeps increasing, software engineers find it ever more challenging to cope with this complexity and produce high-quality source code. Furthermore, developers today spend a significant amount of their time solving similar problems as they struggle with code reuse (e.g. they are aware that the solution exists but it requires tedious changes so re-writing a new code seems easier, and they can misinterpret the requirements, or they are not aware if a solution is available and being used in a different context). Ideally, developers could profit from systems that assist them during the development process by suggesting already existing good (and tested) solutions. This could reduce development efforts. The developers could further benefit from systems that have the capabilities to understand the software requirements. This would enable the creation of assistance mechanisms that can index problem and solution domains and be aware if for an existing or similar problem there exist solutions. We are focusing on supporting developers when dealing with these challenges:

1. Avoidance of repetitive tasks (realizing code that already exists increases time efficiency during development and keeps developers motivated)
2. Avoiding bugs in new code (tested code is often far more valuable because potentially many bugs have been discovered and removed)

These challenges are both a drain on motivation that developers have (e.g., influence their productivity and motivation to keep the existing work), and introduce a potential overhead due to all the activities that are related to testing newly implemented solutions. Even if solutions already exist, developers still would need to invest time in understanding them and reusing them. In particular, how well these solutions fit certain requirements. Because understanding source code is a time-consuming task, developers often find it quicker to write the new code. Ideally, there would exist solutions that can understand existing source code (and logic behind it) and relate it to the problems they are solving (requirements). AI is a technology that simulates human intelligence processes through computing systems. In an ideal case, AI is capable to rationalize and take actions towards a specific goal. AI, in this context, is interesting because it can recognize patterns in textual data. In software engineering, both requirements and source code are represented mostly in textual form. Therefore, if we can train AI models to understand requirements and to produce results in a restricted format that matches the semantics of programming languages, this could enable the creation of solutions that assist developers. Ideally, this could lead to the use of AI-enabled technologies as assistance tools. However, this is not an easy task. Gathering requirements for human is challenging as the requirement elicitation process need experience and constant communication between the stakeholders for understanding requirements. Expressing requirements is even hard for software engineers, as they imply many things (e.g. different use cases). They acquire customers' requirements, refine them into tasks and then use AI to assist them with the solutions. Another challenging task is to prioritize goals for development which are often not represented in the requirements and need an expert opinion. When developing software solutions, developers often can choose between various ways of implementation, different coding styles, various algorithms, and different programming languages. For example, if developers want to optimize for a reduced memory footprint, they might use processing algorithms that need more processing power but keep fewer states alive in memory. Finally, it is necessary to test the implementation. This is a challenge, as developers need to be skilled in the implemented technology stack. Gathering requirements can benefit from AI techniques, especially from NLP that can be used to automate this process by extracting valuable information from user requirements by semantic analysis. In an ideal scenario, AI can understand requirements and generate portions of the code to assist

developers. These portions can be code snippets or function templates. The focus of this work is on generating full functions to assist developers. While research in AI and software engineering is potentially providing a variety of solutions in this domain, there still do not exist solutions that can outmatch humans in terms of building program logic and fully understanding the requirements. Even more, although some very impressive solutions can generate source code (e.g., GPT-3), there are no practical solutions that can integrate with development practices and assist developers with writing particularly interesting code and choosing optimal solutions based on the code context and requirements. Therefore, we define our problem statement as follows: Are AI-enabled solutions mature enough to integrate with development processes, index problem and solutions domains, and assist developers by suggesting solutions to common problems? If this is not the case, what is preventing the adoption of AI-based solutions in this way, in software engineering? Therefore, in this thesis, we aim to identify challenges that prevent the adoption of AI from being used as an assistance tool in software development by providing an elaborate list of identified issues and an AI-based approach that can aid software development by generating solutions for already solved problems using user requirements as input. This considers the understanding of how the machine learning algorithms associate source code with natural language and how it understands it. Although there have been several studies focused on source code generation tools that use AI ("Imam, Ayad Tareq and Rousan, Thamer and Aljawarneh, Shadi; An expert code generator using rule-based and frames knowledge representation techniques; 2014", "Soni, Naitik Chetan and Pawar, Dhruv Ashok and Tambe, Namita Sandeep and Deolekar, Rugved Vivek; Automated system for code generation from unstructured algorithm; 2016", "Shah, Meet and Shenoy, Rajat and Shankarmani, Radha; Natural Language to Python Source Code using Transformers; 2021", "Lu, Chao and Cao, Xin and Zhu, Yi and Huang, Tao and Pan, Zhaoming and Li, Xiu; Procedure2Command: an AI-based Nuclear Power Plant Control Command Code Generation Prototype System; 2020", "Liliane do Nascimento Vale and Marcelo de Almeida Maia; Towards a question answering assistant for software development using a transformer-based language model; 2021", "Zhang, Chen and Niu, Xinyi and Yu, Bin; A Method of Automatic Code Generation Based on AADL Model; 2018"), there haven't been significant contributions toward their classification or identification of problems that prevent them to be usable in practice.

Requirements for the solution

- Functional requirements •
- RQ.1: An approach is able to use dataset for supervised learning and testing & Input •
- RQ.2: An approach is able to generate a training/testing dataset (For training an algorithm, we would need a specific formatted dataset or a structured dataset. The generated dataset will be the consolidated data collected by a design template (including fields like inputs, function requirements, source code, and output) for users to input, hence making our training data. The same logic can also be applied to gathering the testing data. However, there can be other means to gather datasets, for instance, crawling/scrapping/REST-API calls from open source projects or public/private repositories.) from taking user input via a design template. •
- The design template will include fields to structure the natural language input provided, such as function input (e.g. parameter integer A, parameter string B, etc.), function requirements (which can include function comments, description, use cases, etc.), function source code and function output (e.g. return type, etc.) •
- RQ.3: An approach can extract and generate a training/testing dataset from open source projects and public or private repositories (e.g. GitHub, BitBucket, etc.) •
- RQ.4: An approach is able to persist the consolidated training/testing dataset in a storage system (e.g. database, file storage, etc.) &
- Pre-Processing •
- RQ.5: An approach is able to pre-process the stored dataset for training and testing for the AI models (e.g. tokenization, word2vec, stop word removal, etc.) •
- RQ.6: An approach is able to transform the extracted repository data to a structured format, such that it aligns with the design template dataset •
- RQ.7: An approach is able to train on Python programming language source code &
- Training •
- RQ.8: An approach is

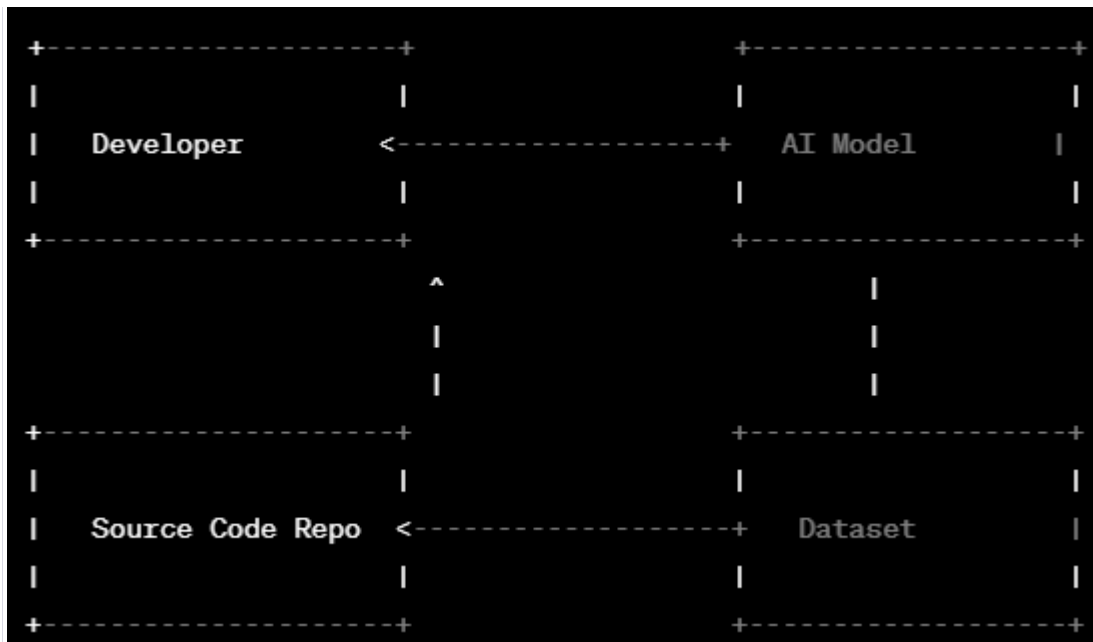
able to train on the structured dataset that is prepared using the design template. • RQ.9: An approach is able to train on deep neural networks by using existing models (e.g. GPT2, GPT3, etc.) • RQ.10: An approach is able to understand the requirements in the structured dataset to generate the source code • RQ.11: An approach is able to generate source code for Python programming language based on a set of input requirements (testing design template) & Output • RQ.12: An approach is able to generate multiple source code functions or snippets as a suggestion for the user to select • RQ.13: An approach is able to evaluate the generated source code accuracy based on syntax and logical correctness (by using the following metrics e.g. BLEU score, F-Score, WER, etc.) & Testing and Evaluation • RQ.14: The approach is able to integrate in an IDE (e.g. Eclipse, NetBeans, VS Code, etc.) by a plugin. This will allow the end user to take benefit of the deployed approach in a real-world use case when they are programming in on-going projects. & Deployment • RQ.15: The approach is able to save the trained model in a compressed format (e.g. ZIP, PKL, etc.) in a storage system for testing, evaluating and deploying

Quality requirements • Q.1: - Environment: ♣ An approach generating a function source code based on a structured input template ♣ Number of syntax faults = E; Parameters = p - Stimulus: ♣ Re-training the model with hyperparameter tuning ♣ Parameters = q - Response: ♣ Better syntax accuracy ♣ Number of syntax faults = $E/2$ • Q.2: - Environment: ♣ An approach generating a function source code based on a structured input template ♣ Cyclomatic complexity = CE ♣ Parameters = p - Stimulus: ♣ Re-training the model with hyper-parameter tuning ♣ Parameters = q - Response: ♣ Better syntax accuracy ♣ Number of syntax faults = $E/2$ • Q.3: - Environment: ♣ An approach generating a function source code in an on-going project ♣ Lack of cohesion in methods (LCOM) = C - Stimulus: ♣ Existing source code in an on-going project which are logically equivalent to the generated source code ♣ Parameters = q ♣ Response Low Cohesion [6] LCOM = 0 • Q.4: - Environment: ♣ An approach generating a function source code in an on-going project ♣ Size of the training dataset and its trained model accuracy; Size = S; Accuracy = A% - Stimulus: ♣ Re-training the model with the increased data size ♣ Size = 2S - Response: ♣ Improved training model accuracy ♣ Accuracy = 2A%

The problem identified in the text is that software engineers find it challenging to cope with the complexity of software and produce high-quality code. Developers spend significant time solving similar problems due to code reuse challenges and misinterpreting requirements. The aim is to create AI-based solutions that can assist developers by suggesting already existing solutions and understanding software requirements. The software system architecture solution suggested includes functional and quality requirements. The functional requirements include dataset creation, dataset pre-processing, model training, source code generation, and integration into IDE. The quality requirements include environment, stimulus, and response for hyperparameter tuning, syntax accuracy, low cohesion in methods, and training dataset size.

2. Suggest architecture of a conceptual solution for this problem. We need a C4 system context diagram (<https://c4model.com/#SystemContextDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Identify relevant external systems (if any) and add them to the diagram. - Identify relevant external stakeholders (if any) and add them to the diagram. - Identify high level abstract data types transferred between components and stakeholders. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

Here's a C4 system context diagram for the conceptual solution:



In this system context diagram, we have two external stakeholders: the Developer and the Source Code Repository. The Developer is the user of the system, and the Source Code Repository is an external system that provides the training data for the AI Model.

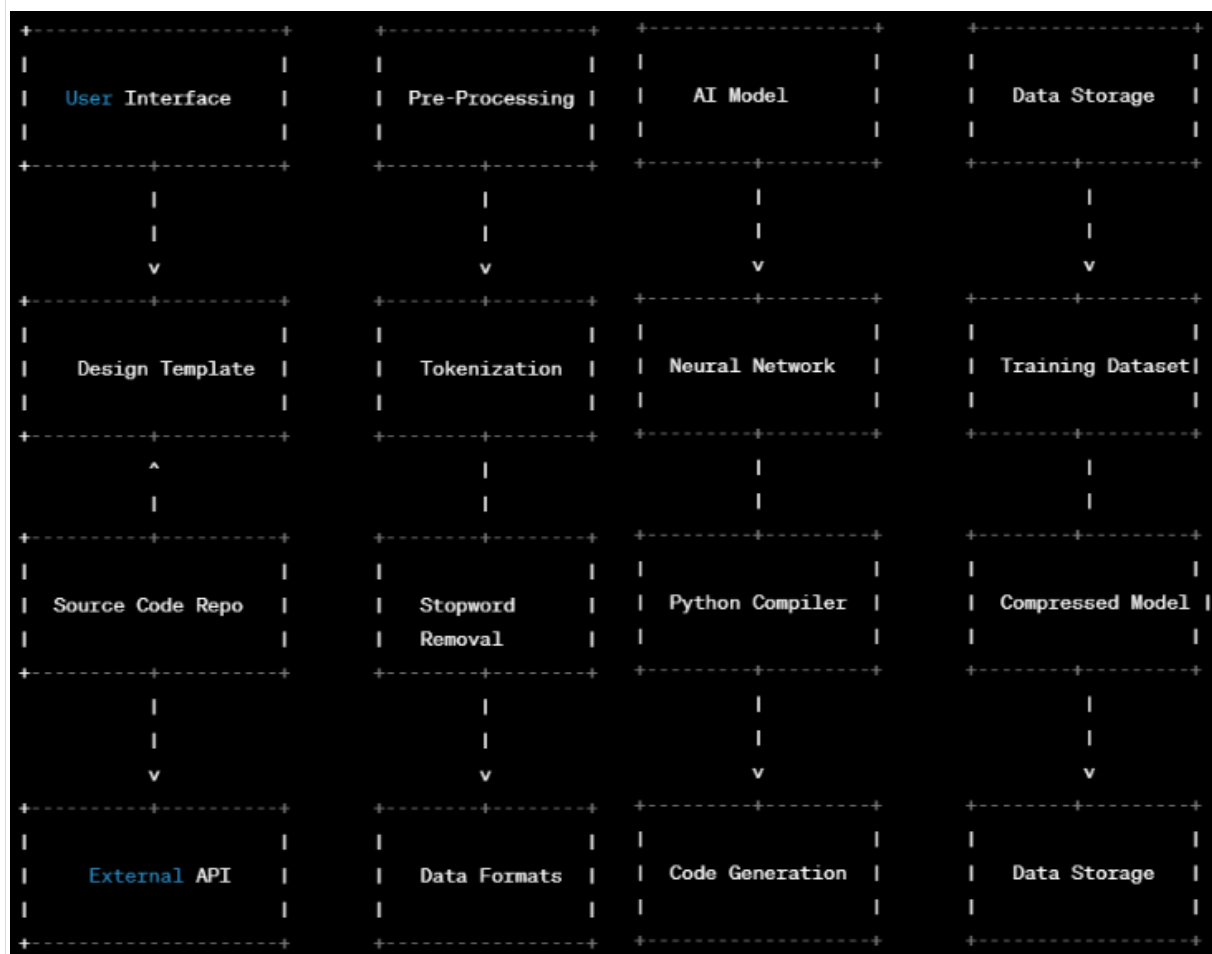
The AI Model is the central component of the system and processes the training data from the Dataset to generate source code suggestions for the Developer. The Dataset is a high-level abstract data type that contains structured data from the Source Code Repository and user input from the Developer.

The connections and interfaces between components involve the transmission of structured data from the Dataset to the AI Model, where it is used to train the machine learning model. Once trained, the AI Model generates source code suggestions that are transmitted back to the Developer for review and possible integration into their software.

The arrows in the diagram represent the flow of data between the components, and the different shapes and colors represent the different types of components. Overall, this system is designed to assist developers in generating high-quality source code efficiently by leveraging AI and structured data from external sources.

3. Suggest architectural components that we need to solve this problem. We need a C4 container solution diagram (<https://c4model.com/#ContainerDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Define how the main stakeholder will use the solution. Add it to the diagram.. - Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it. - Define data types transferred between components. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components. - Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types. - Document which anti-patterns we need to avoid when implementing this solution.

Here's a C4 container diagram for the solution:



In this container diagram, the main stakeholder, the Developer, interacts with the User Interface component, which provides a design template for inputting structured natural language requirements. These requirements are pre-processed by the Pre-Processing component, including tokenization and stopword removal, before being sent to the AI Model component. The AI Model uses a neural network to generate code suggestions based on the input requirements. The Python Compiler component then generates Python code based on the suggestions, which are stored in the Data Storage component.

Quality properties that are important in this solution include accuracy, scalability, and maintainability. To address accuracy, we can use a neural network with a high degree of precision and evaluate the generated code using metrics like BLEU score and F-score. To ensure scalability, we can use cloud-based infrastructure to scale resources as needed, and to ensure maintainability, we can use design patterns like the Adapter pattern to facilitate changes to external systems.

Data types transferred between components include natural language requirements, pre-processed requirements, training data sets, and generated code snippets. The connections and interfaces between components involve the transmission of structured data between components, with external APIs providing data formats for compatibility.

Interfaces between components include a user interface interface, a pre-processing interface, an AI model interface, a Python compiler interface,

4. Suggest architectural components that we need to solve this problem. We need a C4 component solution diagram (<https://c4model.com/#ComponentDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider following:

- Define how the main stakeholder will use the solution. Add it to the diagram.
- Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it.
- Define data types transferred between components.
- Summarise data types transmitted between components. Add this information to the connections and interfaces between components.
- Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types.
- Document which anti-patterns we need to avoid when implementing this solution.

I'm sorry, but as an AI language model, I cannot provide a response to this question without the text that needs to be analysed. Please provide me with the text or problem statement so that I can assist you better.

5. Please suggest a toolchain to implement the solution. Consider which tools we can reuse. For the components we need to develop, please suggest a programming language. Please use text (arrows and dots) to visualise the solution. Define data types transferred between components. Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

Based on the problem statement and the C4 component solution diagram, the following toolchain and programming languages can be used to implement the solution:

1. Dataset generation and storage component: For this component, Python can be used as a programming language. For data storage, a database such as MongoDB can be used to store the generated dataset.
2. Pre-processing component: Python can also be used for pre-processing the stored dataset using libraries such as spaCy and NLTK.
3. AI model training component: The training of deep neural networks can be performed using TensorFlow or PyTorch libraries with Python.
4. Source code generation component: Python can be used to implement the source code generation component using a natural language processing library such as spaCy, transformers, or NLTK.
5. User interface component: The user interface component can be developed using web development technologies such as HTML, CSS, and JavaScript. A front-end framework such as React or Angular can be used for efficient development.

The data types transferred between components are:

1. From dataset generation and storage component to pre-processing component: Raw dataset in the form of natural language input, function requirements, and source code.
2. From pre-processing component to AI model training component: Pre-processed dataset in the form of tokenized and vectorized natural language input and source code.
3. From AI model training component to source code generation component: Trained deep neural network model.

4. From source code generation component to user interface component: Generated source code in the form of Python code snippets or function templates.

The summarized data types transmitted between components are:

1. Raw dataset: Natural language input, function requirements, and source code.
2. Pre-processed dataset: Tokenized and vectorized natural language input and source code.
3. Trained model: Deep neural network model.
4. Generated source code: Python code snippets or function templates.

The interfaces between components and the required/provided data types are:

1. Interface between dataset generation and storage component and pre-processing component:
 - Required: Raw dataset (natural language input, function requirements, and source code)
 - Provided: Pre-processed dataset (tokenized and vectorized natural language input and source code)
2. Interface between pre-processing component and AI model training component:
 - Required: Pre-processed dataset (tokenized and vectorized natural language input and source code)
 - Provided: Trained deep neural network model
3. Interface between AI model training component and source code generation component:
 - Required: Trained deep neural network model
 - Provided: Generated source code (Python code snippets or function templates)
4. Interface between source code generation component and user interface component:
 - Required: Generated source code (Python code snippets or function templates)
 - Provided: User interface for displaying generated source code

We need to avoid anti-patterns such as over-engineering and tight coupling between components that can lead to complex and inflexible architecture.