

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

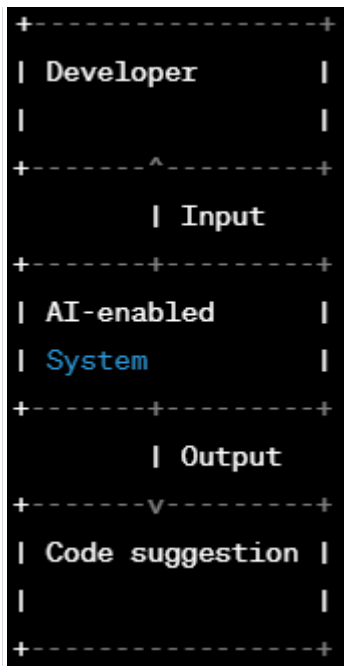
able to train on the structured dataset that is prepared using the design template. • RQ.9: An approach is able to train on deep neural networks by using existing models (e.g. GPT2, GPT3, etc.) • RQ.10: An approach is able to understand the requirements in the structured dataset to generate the source 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 described is the challenge of coping with increasing software complexity and the resulting drain on developers' motivation and productivity. Specifically, the problem is the difficulty of avoiding repetitive tasks and avoiding bugs in new code. The suggested solution is an AI-enabled system that assists developers by suggesting existing solutions to problems and generating new code based on natural language input. The proposed software system architecture should be able to use a structured input template, generate and pre-process training and testing datasets, extract data from open source projects and repositories, train on Python programming language source code using deep neural networks, understand structured requirements and generate source code, evaluate the accuracy of generated code, and integrate with an IDE. The quality requirements of the system include better syntax accuracy, reduced cyclomatic complexity, improved low cohesion in methods, and improved model accuracy with increased data 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 is a C4 system context diagram for the proposed AI-enabled system that assists developers in software development:



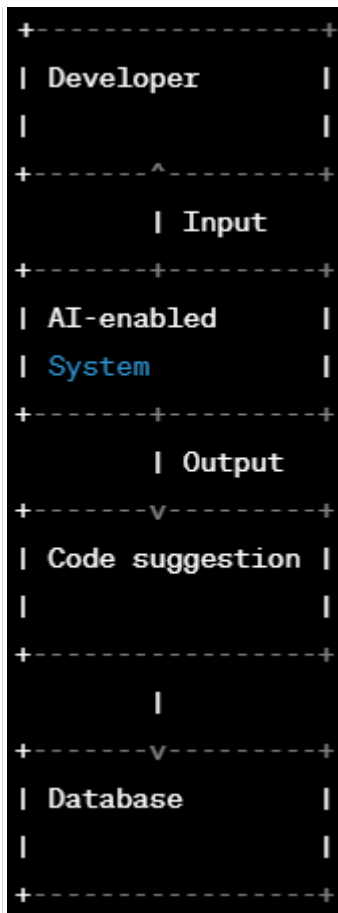
In this diagram, the external stakeholder is the Developer who inputs natural language requirements into the AI-enabled system. The AI-enabled system receives the input, processes it, and generates code suggestions as output.

The data types transmitted between the Developer and the AI-enabled System are structured natural language requirements, while the data types transmitted between the AI-enabled System and the Code Suggestion component are generated code snippets or function templates.

There are no external systems in this diagram as the AI-enabled system is a standalone solution.

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 is a C4 container diagram for the proposed AI-enabled system that assists developers in software development:



In this diagram, the main stakeholder, Developer, interacts with the AI-enabled System by providing structured natural language requirements as input. The AI-enabled System processes the input, generates code suggestions as output, and stores the training and testing datasets in a Database.

The important quality properties for this solution are accuracy, performance, and maintainability. To address these, the architectural components include:

- Natural Language Processing (NLP) Engine: responsible for processing the input and extracting relevant information from the structured natural language requirements.
- Training and Testing Data Generation Component: responsible for generating training and testing datasets from user input or open source projects and repositories.
- Pre-Processing Component: responsible for pre-processing the training and testing datasets for the AI models, including tokenization and stop word removal.
- Deep Neural Network Component: responsible for training on the pre-processed datasets and generating source code suggestions based on the structured natural language requirements.
- Code Quality Evaluation Component: responsible for evaluating the syntax and logical correctness of the generated source code.
- User Interface Component: responsible for providing an interface for the Developer to input requirements and receive code suggestions.
- Database Component: responsible for storing the consolidated training and testing datasets.

The data types transferred between components are structured natural language requirements, training and testing datasets, and generated code snippets or function templates.

The interfaces between components include:

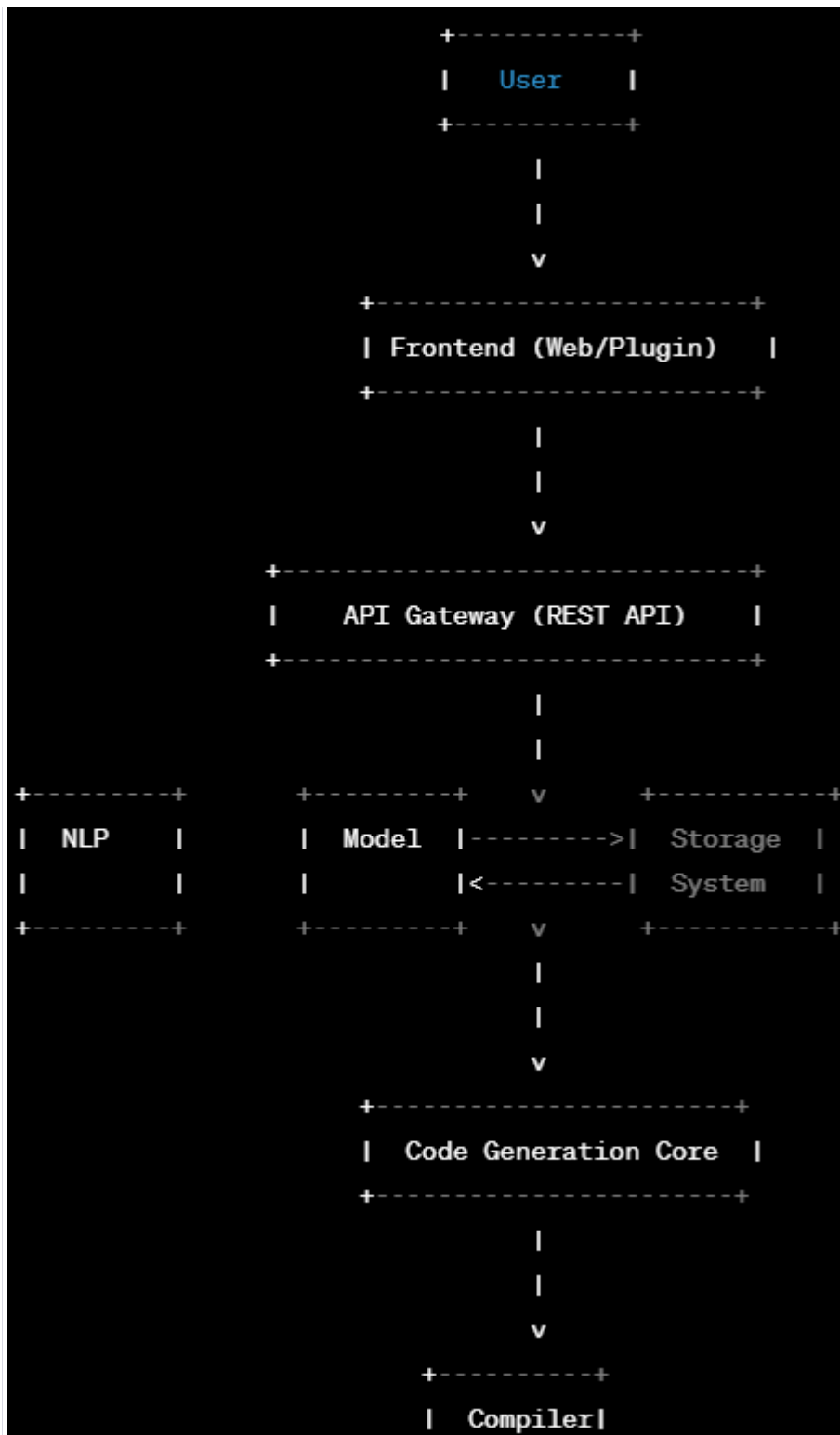
- The User Interface Component requires the NLP Engine and the Code Suggestion Component, and provides structured natural language requirements as input.
- The NLP Engine requires the Pre-Processing Component and provides extracted relevant information from the structured natural language requirements as output.
- The Training and Testing Data Generation Component requires input from the User Interface Component or from open source projects and repositories, and provides the consolidated training and testing datasets as output.
- The Pre-Processing Component requires the training and testing datasets as input and provides pre-processed datasets as output.
- The Deep Neural Network Component requires the pre-processed training and testing datasets as input and provides generated code suggestions as output.
- The Code Quality Evaluation Component requires generated code suggestions as input and provides evaluation metrics as output.
- The Database Component provides storage for the consolidated training and testing datasets.

To avoid anti-patterns, the components should be decoupled and cohesive, with well-defined interfaces between them. The architecture should be modular and extensible, allowing for easy updates and improvements. Additionally, best practices for AI model training and evaluation should be followed to ensure accuracy and maintainability.

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.

As an AI-enabled solution to assist developers with code generation, the following C4 component solution diagram is suggested:



Main stakeholder: Developers

The developers can interact with the solution using a web interface or a plugin in their IDE. They can provide the requirements for the code they want to generate and select from the generated solutions.

Quality properties:

- Accuracy: The generated code should be syntactically and semantically correct.
- Scalability: The system should be able to handle a large amount of data and concurrent user requests.
- Modifiability: The system should be easy to modify and extend with new functionalities.

Components and their relationships:

- Frontend: The frontend component is responsible for handling the user interface and collecting the user requirements. It communicates with the API Gateway component to pass on the user requirements for generating code.
- API Gateway: The API Gateway component acts as a gateway to the backend services, exposing a REST API for the frontend to interact with. It receives the user requirements and passes them on to the NLP and Model components for processing. It also handles authentication and authorization for user requests.
- NLP: The NLP component is responsible for processing the natural language requirements provided by the user and extracting relevant information from them. It communicates with the Model component to generate the code snippets based on the processed requirements.
- Model: The Model component is responsible for generating the code snippets based on the processed requirements. It uses deep neural network models such as GPT-3 for generating code snippets based on the requirements.
- Storage System: The storage system component is responsible for storing the training data, the trained models, and the generated code snippets.
- Code Generation Core: The code generation core component is responsible for taking the processed requirements from the NLP component and generating the code snippets using the models from the Model component. It communicates with the Compiler component to compile the generated code snippets.
- Compiler: The Compiler component is responsible for compiling the generated code snippets and producing the final executable code.

Data types:

- User requirements (natural language text)
- Processed user requirements (structured data)
- Code snippets (structured data)

Interfaces:

- Frontend -> API Gateway: User requirements (natural language text)
- API Gateway -> NLP: User requirements (natural language text)
- NLP -> Model: Processed user requirements (structured data)
- Model -> Code Generation Core: Generated code snippets (structured data)
- Code Generation Core -> Compiler: Generated code snippets (structured data)

Anti-patterns to avoid:

- Tight Coupling: The components should be loosely coupled to allow for easy modifications and replacements without affecting the entire system.

- God Object: The system should not rely on a single component to perform all the tasks. The responsibilities should be distributed among different components to improve modifiability and scalability.
- Over-Engineering: The system should not be over-engineered with unnecessary functionalities that

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 proposed solution architecture, we can suggest the following toolchain to implement the solution:

1. Data Collection:
 - Python-based web scraping tools (e.g., Scrapy, BeautifulSoup, etc.) to extract data from open-source projects and public or private repositories
 - GitHub API or similar REST APIs to extract data from the repositories
2. Data Pre-processing:
 - Python-based NLP libraries (e.g., spaCy, NLTK, etc.) for tokenization, stop-word removal, and word embeddings
 - Python-based libraries for data cleaning and structuring (e.g., Pandas, Numpy, etc.)
3. Machine Learning Models:
 - Python-based Deep Learning libraries (e.g., TensorFlow, PyTorch, etc.) for training the machine learning models
 - Pre-trained models (e.g., GPT-3) for fine-tuning on the specific problem
4. Code Generation:
 - Python-based libraries for code generation from the natural language text (e.g., OpenAI's GPT-3 API, NL2Code, etc.)
 - Python-based libraries for code execution (e.g., CodeRunner, PyPy, etc.)
5. User Interface:
 - An IDE plugin (e.g., for Eclipse, NetBeans, VS Code, etc.) to integrate the solution into the development process
 - A web-based interface (e.g., Flask or Django) to provide an API for the users to interact with the system

The programming language used for implementing the solution can be Python, as it provides excellent support for NLP and Deep Learning libraries.

The data types transferred between the components are primarily textual data, such as requirements, source code, and natural language queries. The structured data includes function inputs, function requirements, function source code, and function outputs. The data is transferred between components using APIs and standard file formats such as JSON or CSV.

The summarized data types transmitted between components are as follows:

- From Data Collection to Data Pre-processing: Raw data in the form of source code files and repositories metadata

- From Data Pre-processing to Machine Learning Models: Structured data in the form of a training dataset, including function inputs, requirements, source code, and output
- From Machine Learning Models to Code Generation: Natural language input as queries or requirements
- From Code Generation to User Interface: Generated code snippets or functions as a suggestion for the user to select

The interfaces between components are defined using REST APIs, allowing communication between the components. The required and provided interfaces and their data types are defined as follows:

- Data Collection:
 - Required Interface: Source code files or repository metadata
 - Provided Interface: Raw data in the form of source code files and repositories metadata
- Data Pre-processing:
 - Required Interface: Raw data in the form of source code files and repositories metadata
 - Provided Interface: Structured data in the form of a training dataset, including function inputs, requirements, source code, and output
- Machine Learning Models:
 - Required Interface: Structured data in the form of a training dataset, including function inputs, requirements, source code, and output
 - Provided Interface: Generated code snippets or functions as a suggestion for the user to select
- Code Generation:
 - Required Interface: Natural language input as queries or requirements
 - Provided Interface: Generated code snippets or functions as a suggestion for the user to select
- User Interface:
 - Required Interface: Generated code snippets or functions as a suggestion for the user to select
 - Provided Interface: An IDE plugin or a web-based interface to provide an API for the users to interact with the system

Some anti-patterns that we need to avoid while implementing the solution are:

- Over-engineering: We should avoid making the system