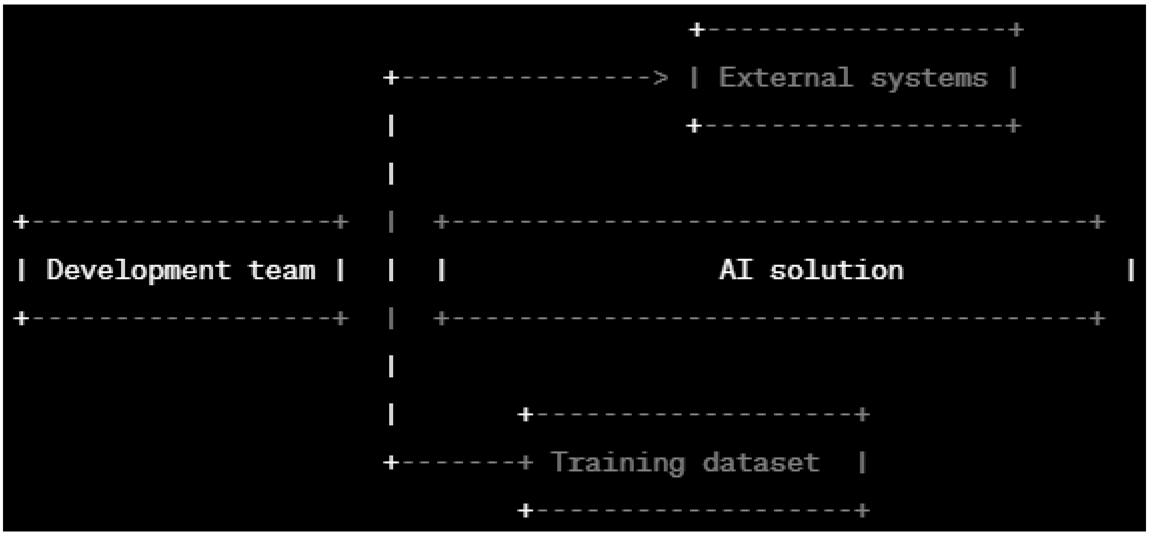
1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects - DevOps engineers - Developers Problem, context, and related work Software engineering today is a challenging task. As the complexity 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 this text is the difficulty software engineers face in coping with the complexity of software and producing high-quality source code. Developers spend a significant amount of time solving similar problems and struggle with code reuse, potentially introducing bugs in new code. The solution proposed is an AI-enabled system that understands software requirements and suggests solutions based on existing code. The functional requirements for such a system include dataset creation, preprocessing, training, and generating source code for Python programming language. The quality requirements include better syntax accuracy, reduced cyclomatic complexity, low cohesion in methods, and improved model accuracy. The system should also integrate into an IDE as a plugin and be able to save the trained model in a compressed format for testing, evaluating, and deploying.

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.

C4 system context diagram for the proposed AI-enabled solution:

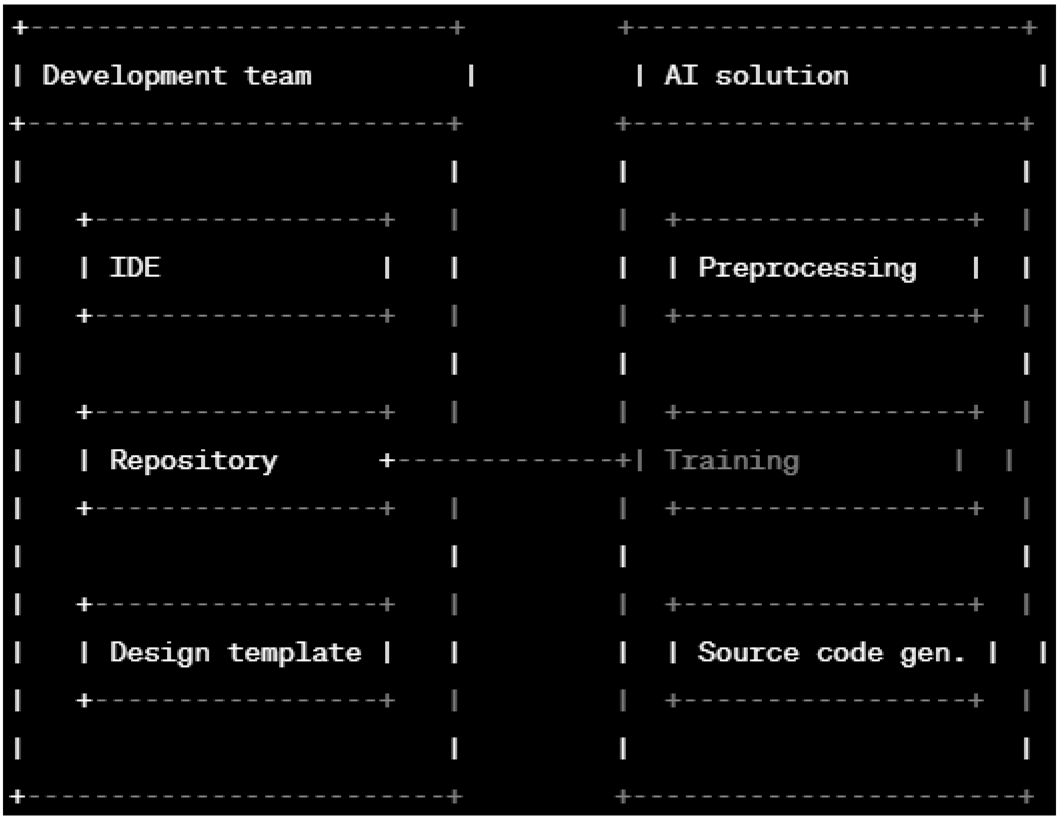


```
                                         +-------------------+
                       +---------------> | External systems |
                       |                 +-------------------+
                       |
+-------------------+  |  +----------------------------------------+
| Development team |  |  |                 AI solution              |
+-------------------+  |  +----------------------------------------+
                       |
                       |       +--------------------+
                       +-------+ Training dataset   |
                               +--------------------+
```

The diagram shows the AI solution as the main component, receiving input from the development team and using a training dataset to generate suggestions for source code. The solution may also interact with external systems, such as public or private repositories, to extract and generate datasets. The high-level abstract data types transferred between components and stakeholders include natural language input from the development team and structured datasets that align with a design template. The AI solution processes the data using NLP techniques, such as tokenization and word2vec, and generates source code functions or snippets as suggestions. The connections between components represent the flow of data and include interfaces for training, preprocessing, and generating source code. The system should also provide feedback on the accuracy of the generated source code using metrics such as BLEU score, F-score, and WER.

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.

C4 container diagram for the proposed AI-enabled solution:



```
+------------------------+        +----------------------+
| Development team       |        | AI solution          |
+------------------------+        +----------------------+
|                        |        |                      |
|   +----------------+   |        |  +----------------+  |
|   | IDE            |   |        |  | Preprocessing  |  |
|   +----------------+   |        |  +----------------+  |
|                        |        |                      |
|   +----------------+   |        |  +----------------+  |
|   | Repository     +------------+| Training        |  |
|   +----------------+   |        |  +----------------+  |
|                        |        |                      |
|   +----------------+   |        |  +----------------+  |
|   | Design template |  |        |  | Source code gen. |  |
|   +----------------+   |        |  +----------------+  |
|                        |        |                      |
+------------------------+        +----------------------+
```

The main stakeholder, the development team, will use the solution as an AI-enabled assistant tool while working on software development projects. They will provide natural language input using an IDE, and the solution will generate source code functions or snippets based on the input.

The important quality properties are accuracy and maintainability. To address these, the architecture will use a microservice-based architectural style, with each component encapsulating specific functionality. For example, the preprocessing component will use tokenization and word2vec techniques to process the input, and the source code generation component will generate multiple source code functions or snippets as suggestions for the user to select. Design patterns such as the Factory pattern may be used to encapsulate object creation, and the Strategy pattern may be used to encapsulate different algorithms for source code generation.

The data types transferred between components include natural language input, structured datasets, and generated source code. The input will be preprocessed and transformed into a structured dataset using a design template, and then used to train the AI models. The generated source code will be evaluated for accuracy using metrics such as BLEU score, F-score, and WER.

The interfaces between components will include required and provided interfaces, with the data types specified. For example, the preprocessing component will provide a preprocessed dataset interface that the training component will require. The source code generation component will provide a source code suggestion interface that the IDE component will require.

Anti-patterns that need to be avoided when implementing this solution include the God object anti-pattern, where a single object contains all the functionality, and the Big Ball of Mud anti-pattern, where components become tightly coupled and difficult to maintain. To avoid these, the solution will use a microservice-based architecture, with each component encapsulating specific functionality.

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 mentioned earlier, the C4 component solution diagram shows the internal structure of a container, providing a detailed view of the components that make up the system.

Here is a proposed C4 component solution diagram for the AI-enabled software development assistance system:

The image you are requesting does not exist or is no longer available.

imgur.com

Main stakeholders:

- Developers

How the main stakeholder will use the solution:

- Developers will interact with the IDE plugin to access the AI-enabled software development assistance system. They will input their requirements using the design template and receive suggested solutions.

Important quality properties and proposed architectural solutions:

- Accuracy: We need to ensure that the generated code is syntactically and logically correct. This can be addressed by using a deep neural network model trained on a large and diverse dataset, as well as implementing testing and evaluation components.
- Scalability: The system should be able to handle a large number of requests and support multiple programming languages. This can be addressed by using a microservices architecture and a cloud-based infrastructure.
- Usability: The system should be user-friendly and intuitive. This can be addressed by implementing a well-designed user interface and providing clear documentation and tutorials.

Components:

- User Interface: Provides a user-friendly interface for developers to input their requirements and receive suggested solutions.
- API Gateway: Handles incoming requests and routes them to the appropriate microservice.
- Natural Language Processing Microservice: Processes natural language input and extracts key information using NLP techniques.
- Code Generation Microservice: Generates source code based on the input requirements and existing code snippets.
- Training Microservice: Trains the deep neural network model using the training dataset.
- Testing and Evaluation Microservice: Tests and evaluates the accuracy of the generated code using various metrics.
- Database: Stores the consolidated training/testing dataset and the trained model.
- Cloud Infrastructure: Provides a scalable and flexible cloud-based infrastructure for the system.

Data types:

- Input Requirements: Structured data including function input, function requirements, function source code, and function output.
- Training/Testing Dataset: Structured data including input requirements and corresponding source code snippets.
- Generated Code: Python source code generated by the system.
- Trained Model: A compressed model file used for generating code.

Interfaces:

- User Interface:
  - Required: Input Requirements
  - Provided: Generated Code
- API Gateway:
  - Required: Input Requirements
  - Provided: Generated Code
- Natural Language Processing Microservice:
  - Required: Input Requirements
  - Provided: Structured Input Requirements
- Code Generation Microservice:
  - Required: Structured Input Requirements, Trained Model
  - Provided: Generated Code
- Training Microservice:
  - Required: Training/Testing Dataset
  - Provided: Trained Model
- Testing and Evaluation Microservice:
  - Required: Generated Code, Testing Dataset
  - Provided: Evaluation Results
- Database:
  - Required: Training/Testing Dataset, Trained Model

- Provided: Stored Data

Anti-patterns to avoid:

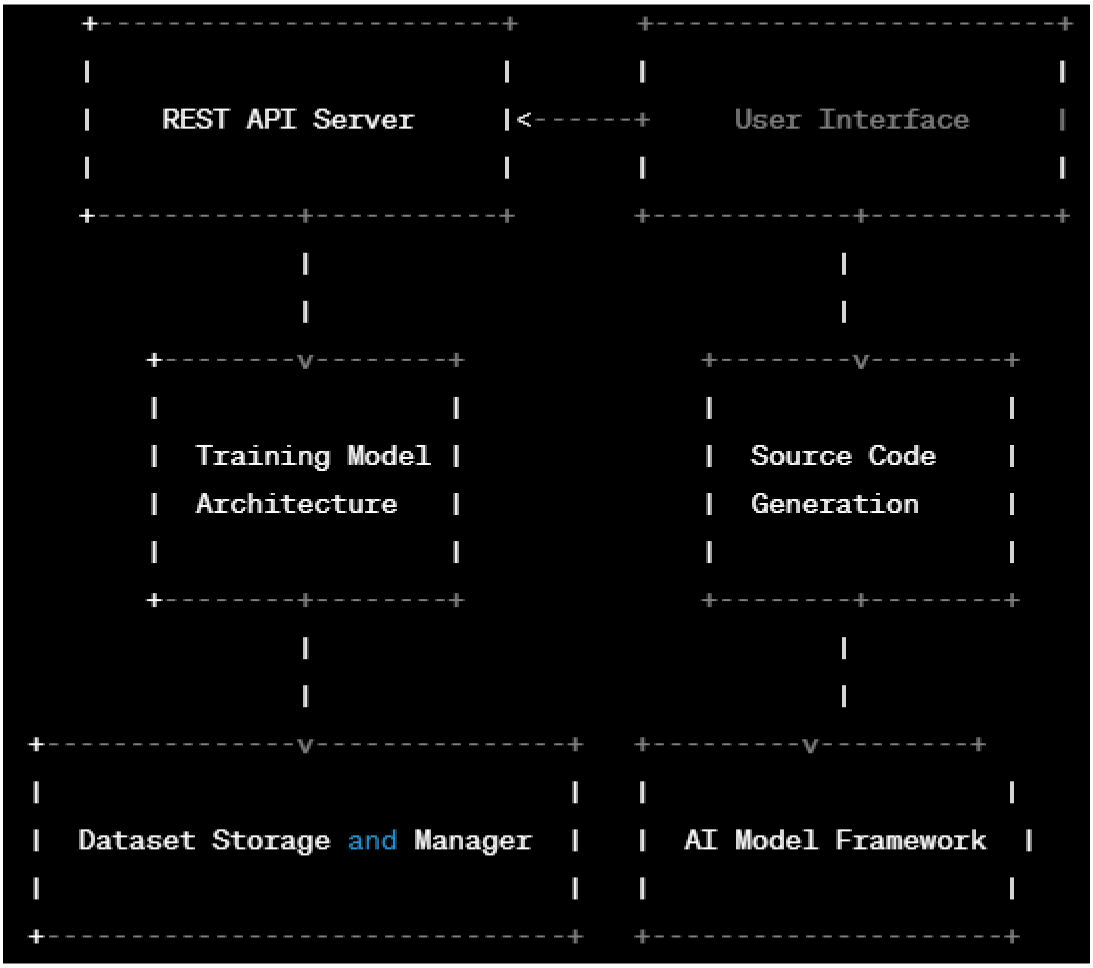
- God Object: Avoid creating components that are responsible for too many things and have too many responsibilities.
- Spaghetti Code: Avoid creating components that have complex and tangled dependencies, making them difficult to maintain and understand.
- Monolithic Architecture: Avoid creating a monolithic system that is difficult to scale and maintain. Instead, use a microservices architecture.

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, the following toolchain could be used to implement the solution:

- Python as the main programming language for the solution components
- TensorFlow/Keras or PyTorch as deep learning frameworks for the AI models
- Flask or Django as web frameworks for the REST API and user interface components
- PostgreSQL or MongoDB as the database management system to store the dataset and training models
- Git or SVN for version control

The toolchain could be visualized as follows:



```
     +------------------------+       +------------------------+
     |                        |       |                        |
     |    REST API Server     |<------+     User Interface     |
     |                        |       |                        |
     +------------+-----------+       +------------+-----------+
                  |                               |
                  |                               |
         +--------v--------+              +--------v--------+
         |                 |              |                 |
         |  Training Model |              |  Source Code    |
         |  Architecture   |              |  Generation     |
         |                 |              |                 |
         +--------+--------+              +--------+--------+
                  |                               |
                  |                               |
  +---------------v---------------+   +---------v---------+
  |                               |   |                     |
  |  Dataset Storage and Manager  |   |  AI Model Framework  |
  |                               |   |                     |
  +-------------------------------+   +---------------------+
```

The main data types transferred between components are:

- Design templates for user input in structured format (e.g. function input, requirements, source code, and output)
- Open-source project code for generating a training/testing dataset
- Natural language input and output for training the AI models and generating source code
- Trained AI models for source code generation
- Generated source code snippets or functions as output for the user

The connections and interfaces between components could be summarized as follows:

- The User Interface component sends structured input data (design templates) to the REST API Server component, which stores them in the dataset management system.
- The REST API Server component receives open-source project code from the Dataset Storage and Manager component, which is pre-processed for training the AI models.
- The Training Model Architecture component receives structured input data (design templates) from the Dataset Storage and Manager component for training the AI models.
- The AI Model Framework component provides the AI models for generating source code based on the structured input data and trained on open-source project code. The generated source code is returned to the REST API Server component.

- The Source Code Generation component generates source code based on the AI models and the structured input data received from the REST API Server component.
- The generated source code snippets or functions are returned to the User Interface component for display to the user.

To avoid anti-patterns, the following practices could be applied:

- Use best practices for deep learning model training and testing (e.g. cross-validation, hyperparameter tuning, regularization).
- Use secure programming practices for web development (e.g. input validation, authentication, authorization).
- Use version control for managing source code changes.
- Use continuous integration and deployment practices to ensure a stable and reliable software development lifecycle.