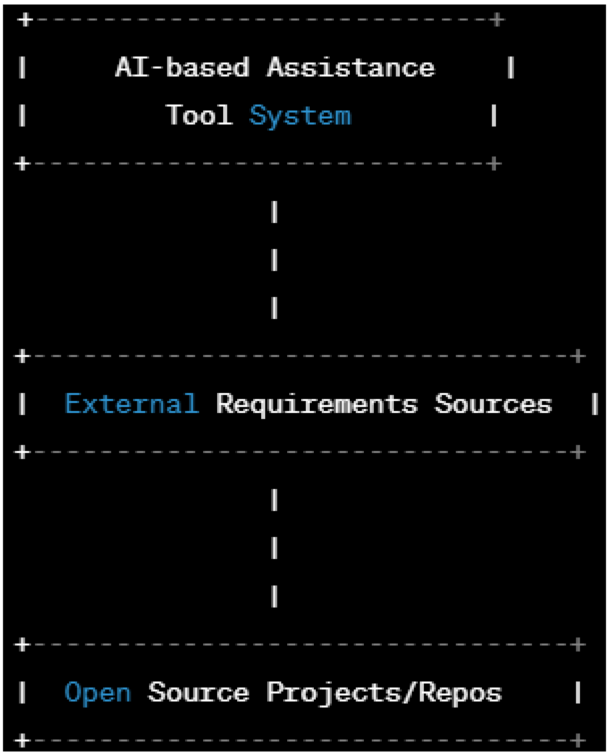
1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects - DevOps engineers - Developers Problem, context, and related work Software engineering today is a challenging task. As the complexity 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 addressed in this text is the challenges that software engineers face when dealing with the complexity of software development and the need for code reuse, which can lead to repetitive tasks and potential bugs in new code. The text suggests that AI can be used to assist developers by suggesting existing solutions to similar problems and understanding software requirements. The proposed software system architecture solution involves a set of functional and quality requirements, including the ability to use a structured dataset for supervised learning and testing, generate a training/testing dataset, extract and generate a dataset from open source projects and repositories, pre-process the dataset, transform extracted repository data to a structured format, train on Python programming language source code, understand requirements to generate source code, generate multiple source code functions or snippets, evaluate the generated source code accuracy, integrate in an IDE, save the trained model in a compressed format, and meet quality requirements. The proposed solution aims to identify challenges preventing the adoption of AI as an assistance tool in software development and provide an AI-based approach that can aid software development by generating solutions for already solved problems using user requirements as input.

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 solution:



```
+---------------------------+
|     AI-based Assistance    |
|        Tool System        |
+---------------------------+
              |
              |
              |
+--------------------------------+
|  External Requirements Sources  |
+--------------------------------+
              |
              |
              |
+--------------------------------+
|  Open Source Projects/Repos    |
+--------------------------------+
```

The system context diagram shows the AI-based Assistance Tool System, which is the proposed solution to the problem described in the text. The external requirements sources and open source projects/repositories are relevant external systems that provide input data for the AI-based Assistance Tool System. The diagram does not show any external stakeholders as they are not relevant to the system context.

The high-level abstract data types that will be transferred between components and systems are as follows:

- Structured input data: This will include structured data from the external requirements sources and open source projects/repositories that will be used to train the AI models.
- AI models: These models will be trained on the structured input data and will be used to generate source code based on user requirements.
- Generated source code: This will be the output of the AI-based Assistance Tool System, which will be provided to the user.

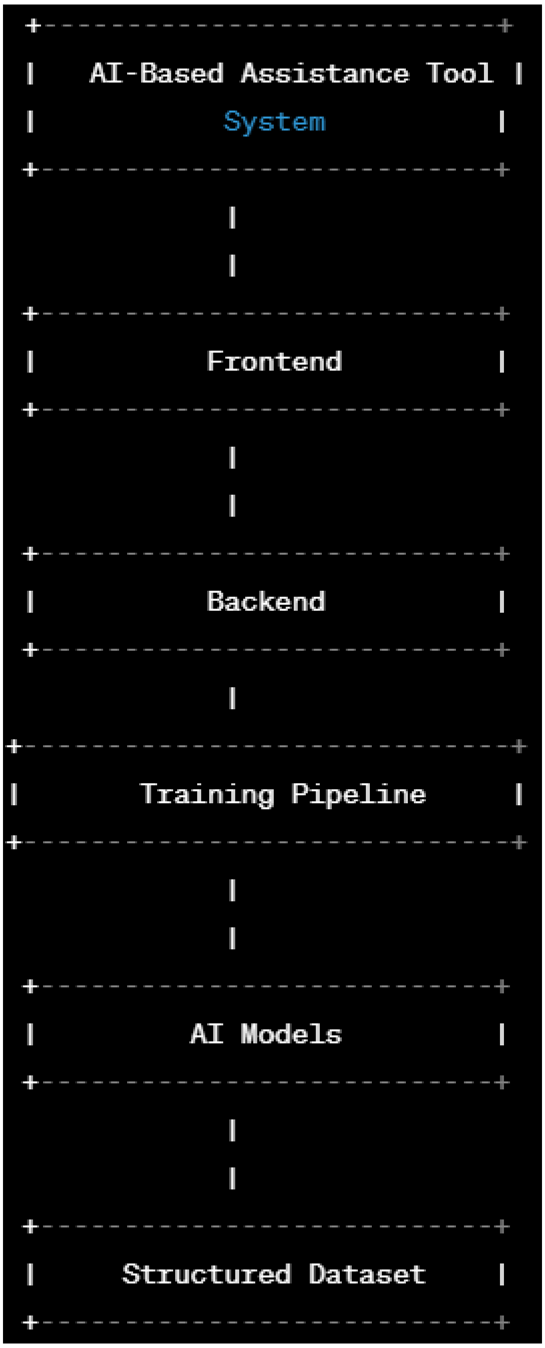
The connections and interfaces between components will involve the transmission of structured input data from the external requirements sources and open source projects/repositories to the AI models, and the transmission of generated source code from the AI models to the user. The AI models will be trained on the structured input data and will use this data to generate source code based on user requirements. The generated source code will be evaluated for syntax and logical correctness, and will be provided to the user as multiple function snippets or templates for them to select.

The connections and interfaces between the components in the diagram are represented by arrows and different shapes. The structured input data is represented by arrows connecting the

External Requirements Sources and Open Source Projects/Repos components to the AI models component. The AI models are represented by a rectangular box. The generated source code is represented by arrows connecting the AI models component to the user. The AI-based Assistance Tool System is represented by a rounded rectangular box. The different colours in the diagram are used to differentiate between the components.

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 solution:



```
+-------------------------+
|  AI-Based Assistance Tool |
|         System          |
+-------------------------+
            |
            |
+-------------------------+
|        Frontend         |
+-------------------------+
            |
            |
+-------------------------+
|        Backend          |
+-------------------------+
            |
+---------------------------+
|     Training Pipeline     |
+---------------------------+
            |
            |
+-------------------------+
|       AI Models         |
+-------------------------+
            |
            |
+-------------------------+
|   Structured Dataset    |
+-------------------------+
```

The main stakeholder will use the solution by interacting with the frontend component of the system. The frontend component will provide a user interface for the stakeholder to input natural language requirements, which will then be processed by the backend component. The backend component will use AI models to generate source code that meets the stakeholder's requirements, which will be displayed in the frontend component.

Important quality properties for the solution are accuracy of generated source code, maintainability of the system, and scalability of the training pipeline. To address these properties, we propose using the following architectural solutions:

- Architectural style: Service-oriented architecture (SOA) to enable modular and scalable development and deployment.

- Design patterns: Model-View-Controller (MVC) for the frontend component to separate concerns and increase maintainability.
- Design tactics: Continuous Integration/Continuous Deployment (CI/CD) to enable rapid and frequent updates to the system, as well as automated testing to ensure accuracy of generated source code.
- Components and their relationships: The frontend component will communicate with the backend component through RESTful API calls. The backend component will consist of the training pipeline component, which will process input data and train AI models, and the AI models component, which will generate source code based on input requirements. The structured dataset component will provide structured input data to the training pipeline component.

The data types transferred between components are as follows:

- Natural language requirements: This will be the input provided by the stakeholder through the frontend component.
- Structured input data: This will be the structured dataset used to train the AI models.
- AI models: These models will be trained on the structured input data and will be used to generate source code based on the natural language requirements provided by the stakeholder.
- Generated source code: This will be the output of the system, which will be provided to the stakeholder through the frontend component.

The connections and interfaces between components are represented by arrows and different shapes. The frontend component is represented by a rectangular box, while the backend component is divided into the training pipeline, AI models, and structured dataset components, each represented by a rectangular box. The connections between the frontend and backend components are represented by arrows. The different colours in the diagram are used to differentiate between the components.

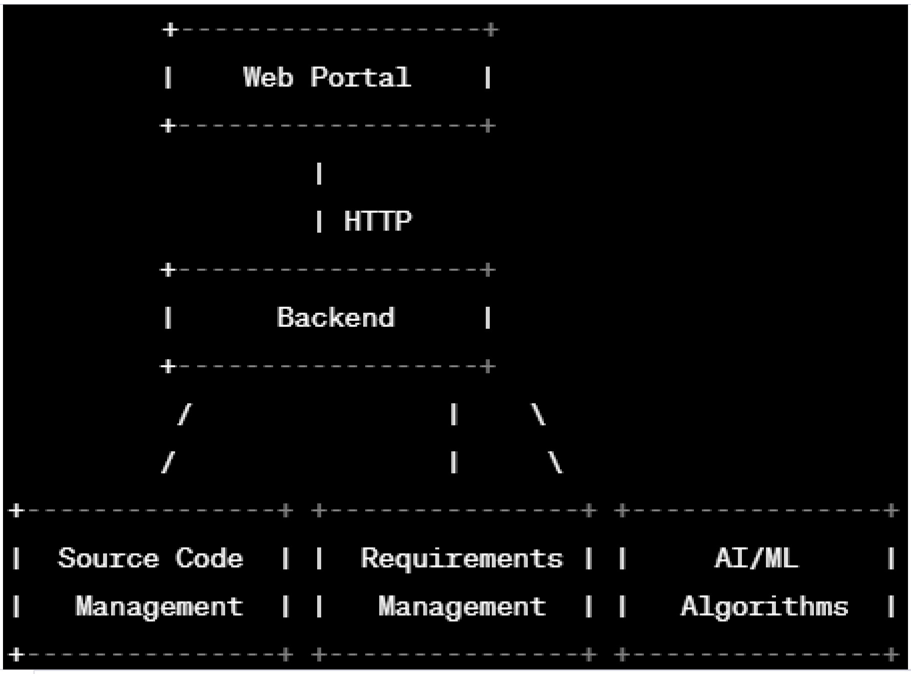
The interfaces between components are as follows:

- Frontend: Provides a RESTful API for the backend component to receive natural language requirements input, and displays generated source code output.
- Backend: Provides a RESTful API for the frontend component to receive natural language requirements input, and provides generated source code output to the frontend component.
- Training pipeline: Provides a RESTful API for the AI models component to receive structured input data, and trains AI models based on this input data.
- AI models: Provides a RESTful API for the backend component to receive natural language requirements input, and generates source code output based on these requirements.
- Structured dataset: Provides structured input data to the training pipeline component.

To avoid anti-patterns, we need to ensure that the system is designed to be scalable and maintainable, and that input and output data

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 requested, here is a suggestion for a C4 component solution diagram:



```
                +-----------------+
                |    Web Portal    |
                +-----------------+
                        |
                        | HTTP
                +-----------------+
                |      Backend     |
                +-----------------+
                 /             |    \
                /              |     \
+--------------+ +--------------+ +--------------+
|  Source Code  | |  Requirements | |     AI/ML     |
|   Management  | |   Management  | |   Algorithms  |
+--------------+ +--------------+ +--------------+
```

- The main stakeholder (developer) will use the solution through a web portal where they can input the requirements and receive suggestions for the code.
- Important quality properties include accuracy of generated code, speed of response, and ease of use for developers. Adequate solutions to address them may include the use of machine learning algorithms, caching to improve response time, and a user-friendly interface. The overall architectural style could be a microservices architecture, with each component being a separate microservice.
- Data types transferred between components may include natural language requirements, structured code templates, and trained machine learning models.
- The connections and interfaces between components can be summarized as follows:
  - The Web Portal sends HTTP requests to the Backend, passing along natural language requirements.
  - The Backend manages the interaction with the Source Code and Requirements Management components and returns suggestions for code to the Web Portal.
  - The Source Code Management component contains a dataset of existing code that can be used as a reference for generating new code snippets.
  - The Requirements Management component contains a dataset of requirements and use cases that can be used to train the machine learning algorithms.

    - The AI/ML Algorithms component contains the machine learning models that can understand the requirements and generate code snippets.
- Interfaces between components could include:
    - The Web Portal requires an HTTP interface to send natural language requirements to the Backend and receive code suggestions in return.
    - The Backend requires access to the Source Code and Requirements Management components to generate code suggestions based on existing code and requirements.
    - The Source Code Management component requires a structured interface to provide the Backend with access to its dataset of existing code snippets.
    - The Requirements Management component requires a structured interface to provide the Backend with access to its dataset of requirements and use cases.
    - The AI/ML Algorithms component requires access to the structured data from the Requirements Management component to train its models.
- Anti-patterns to avoid when implementing this solution could include overcomplicating the machine learning models, not providing enough training data, and ignoring the importance of accuracy in the generated code.

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.

To implement the proposed solution, we can use the following toolchain:

1. Design tool: We can use tools like Figma, Sketch, or Adobe XD to create a design template for the natural language input provided by users.
2. Data collection and storage: We can use web scraping libraries like BeautifulSoup or Scrapy to crawl open-source repositories on GitHub and BitBucket, and collect the structured dataset for training and testing. This data can be stored in a database or file storage system like MongoDB or Amazon S3.
3. Data preprocessing: We can use Python-based NLP libraries like NLTK or spaCy to preprocess the natural language input and convert it into structured data for training the machine learning model.
4. Machine learning model: We can use existing deep learning models like GPT-3 to generate source code from structured data. We can use deep learning libraries like TensorFlow or PyTorch to build and train our own models.
5. Code generation and evaluation: Once the model is trained, we can generate the source code based on the structured data provided by the user. We can use Python to generate and evaluate the code snippets based on syntax and logical correctness.
6. Integration: We can integrate the solution with an IDE by creating a plugin for popular IDEs like Eclipse, NetBeans, or VS Code.

The following programming languages can be used for each component:

1. Design tool: HTML/CSS/JS
2. Data collection and storage: Python
3. Data preprocessing: Python

4. Machine learning model: Python/TensorFlow/PyTorch
5. Code generation and evaluation: Python
6. Integration: Java/Python

Data types transferred between components:

- Natural language input (string)
- Structured data (JSON)
- Source code (string)
- Evaluation metrics (JSON)

Summarised data types transmitted between components:

- Design tool -> Data collection and storage: User input (string)
- Data collection and storage -> Data preprocessing: Structured data (JSON)
- Data preprocessing -> Machine learning model: Structured data (JSON)
- Machine learning model -> Code generation and evaluation: Source code (string)
- Code generation and evaluation -> Integration: Generated code (string), evaluation metrics (JSON)

Interfaces between components:

- Design tool:
  - Required interface: None
  - Provided interface: User input (string)
- Data collection and storage:
  - Required interface: None
  - Provided interface: Structured data (JSON)
- Data preprocessing:
  - Required interface: Natural language input (string)
  - Provided interface: Structured data (JSON)
- Machine learning model:
  - Required interface: Structured data (JSON)
  - Provided interface: Source code (string)
- Code generation and evaluation:
  - Required interface: Structured data (JSON)
  - Provided interface: Generated code (string), evaluation metrics (JSON)
- Integration:
  - Required interface: Generated code (string)
  - Provided interface: None

Anti-patterns to avoid:

- Tight coupling between components
- Over-engineering of the solution
- Lack of scalability in the system