MODEL TRAINING METHOD, CODE PROCESSING METHOD, AND CODE PROCESSING PLATFORM

The present disclosure claims priority to Chinese Patent Application No. 202410384068.8, entitled "MODEL TRAINING METHOD, CODE PROCESSING METHOD, AND CODE PROCESSING PLATFORM" and filed with the China National Intellectual Property Administration on March 29, 2024, which is incorporated herein by reference in its entirety.

TECHNICAL FIELD

Embodiments of the present disclosure relate to the field of computer technologies, and specifically, to a model training method, a code processing method, and a code processing platform.

BACKGROUND

With the continuous development of computer technologies, many organizations such as institutions or enterprises develop software based on the actual requirements, to provide services for themselves or users. During software development, a developer may use a neural network model to help the writing of software code. Written code data is inputted into the neural network model, to obtain a processing result corresponding to the code data.

Before the code data is processed by using the neural network model, model training needs to be performed on the neural network model by using code sample data. However, in the prior art, in a model training process, code samples independent of each other tend to be used for model training. Consequently, performance of a trained neural network model is poor, and code data having a complex relationship in an actual development scenario cannot be well processed. As a result, efficiency of software development is low, and developed software cannot be used in time to provide services. Therefore, there is an urgent need to provide a model with good performance that can process code data having a complex relationship.

SUMMARY

In view of this, embodiments of the present disclosure provide a model training method. One or more embodiments of the present disclosure jointly relate to a code processing method, a model training apparatus, a code processing apparatus, a code processing platform, a computing device, a computer-readable storage medium, and a computer program product, to resolve technical defects in the prior art that a neural network model has poor performance and cannot well process code data that has a complex relationship in an actual development scenario.

According to a first aspect of the embodiments of the present disclosure, a model training method is provided, and includes:

determining a first sample code file and a second sample code file that correspond to a to-be-

trained code processing model, where a call relationship exists between the first sample code file and the second sample code file;

determining first sample code data and a sample code label based on the first sample code file;

determining second sample code data based on the second sample code file, and determining sample code data based on the first sample code data and the second sample code data; and

performing model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model.

According to a second aspect of the embodiments of the present disclosure, a model training apparatus is provided, and includes:

a code file determining module, configured to determine a first sample code file and a second sample code file that correspond to a to-be-trained code processing model, where a call relationship exists between the first sample code file and the second sample code file;

a first sample determining module, configured to determine first sample code data and a sample code label based on the first sample code file;

a second sample determining module, configured to determine second sample code data based on the second sample code file, and determine sample code data based on the first sample code data and the second sample code data; and

a model training module, configured to perform model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model.

According to a third aspect of the embodiments of the present disclosure, a code processing method is provided, and includes:

receiving to-be-processed code data sent by a user through a code processing interface in the client;

inputting the to-be-processed code data into a code processing model, and performing a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data, where the code processing model is obtained through training according to the foregoing model training method, and the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation; and

sending the target code data to the client, such that the client presents the target code data to the user through the code processing interface.

According to a fourth aspect of the embodiments of the present disclosure, a code processing apparatus is provided, and includes:

a code data receiving module, configured to receive to-be-processed code data sent by a user through a code processing interface in the client;

a code data processing module, configured to input the to-be-processed code data into a code processing model, and perform a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data, where the code processing model is obtained through training according to the foregoing model training method, and the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation; and

a code data sending module, configured to send the target code data to the client, such that the client presents the target code data to the user through the code processing interface.

According to a fifth aspect of the embodiments of the present disclosure, a code processing platform is provided, and includes a request receiving unit, a code processing unit, and a response unit, where

the request receiving unit is configured to receive a code processing request, and determine, based on the code processing request, to-be-processed code data and a code processing model identifier corresponding to the to-be-processed code data;

the code processing unit is configured to determine a code processing model based on the code processing model identifier, input the to-be-processed code data into the code processing model, and perform a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data, where the code processing model is obtained through training according to the foregoing model training method, and the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation; and

the response unit is configured to output the target code data.

According to a sixth aspect of the embodiments of the present disclosure, a computing device is provided, and includes:

a memory and a processor, where

the memory is configured to store a computer program/computer instructions, the processor is configured to execute the computer program/ computer instructions, and when the computer program/ computer instructions are executed by the processor, steps of the foregoing model training method or steps of the foregoing code processing method are implemented.

According to a seventh aspect of the embodiments of the present disclosure, a computer-readable storage medium is provided. The computer-readable storage medium stores a computer program/computer instructions, and when the computer program/computer instructions are executed by a processor, steps of the foregoing model training method or steps of the foregoing

code processing method are implemented.

According to an eighth aspect of the embodiments of the present disclosure, a computer program product is provided. The computer program product includes a computer program/computer instructions, and when the computer program/computer instructions are executed by a processor, steps of the foregoing model training method or steps of the foregoing code processing method are implemented.

According to the model training method in one or more embodiments of the present disclosure, a first sample code file and a second sample code file that has a call relationship with the first sample code file may be determined, and model training may be performed based on first sample code data and a sample code label that are determined based on the first sample code file, and second sample code data determined based on the second sample code file. In this way, in a model training process, a code processing model is trained by using sample code files that have the call relationship, so that a trained code processing model can process code data having a complex call relationship in an actual development scenario, thereby improving performance of the code processing model in the actual scenario. In addition, when the code processing model having relatively good performance is applied to the actual development scenario, efficiency of software development can be improved, and software development can be efficiently completed, thereby avoiding a problem that developed software cannot be used in time to provide services.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a schematic diagram of an application of a data processing method according to an embodiment of the present disclosure;

FIG. 2 is a flowchart of a model training method according to an embodiment of the present disclosure;

FIG. 3 is a schematic diagram of a sample code file of a model training method according to an embodiment of the present disclosure;

FIG. 4 is a schematic diagram of a prompt of a model training method according to an embodiment of the present disclosure;

FIG. 5 is a flowchart of a processing process of a model training method according to an embodiment of the present disclosure;

FIG. 6 is a flowchart of a code processing method according to an embodiment of the present disclosure;

FIG. 7 is a schematic structural diagram of a code processing platform according to an embodiment of the present disclosure;

FIG. 8 is a schematic structural diagram of a model training apparatus according to an

embodiment of the present disclosure;

FIG. 9 is a schematic structural diagram of a code processing apparatus according to an embodiment of the present disclosure; and

FIG. 10 is a structural block diagram of a computing device according to an embodiment of the present disclosure.

DETAILED DESCRIPTION

In the following descriptions, many specific details are described for fully understanding the present disclosure. However, the present disclosure can be implemented in other manners different from those described herein. A person skilled in the art may make similar generalization without departing from connotation of the present disclosure. Therefore, the present disclosure is not limited by following disclosed specific embodiments.

Terms used in one or more embodiments of the present disclosure are merely used to describe specific embodiments, and are not intended to limit the one or more embodiments of the present disclosure. Terms “a/an/one”, “the”, and “this” of singular forms used in the embodiments and the appended claims of the present disclosure are also intended to include plural forms, unless otherwise specified in the context clearly. It should further be understood that the term “and/or” used in one or more embodiments of the present disclosure indicates and includes any or all possible combinations of one or more associated listed items.

It should be understood that although terms such as “first” and “second” may be used to describe various types of information in one or more embodiments of the present disclosure, such information should not be limited by these terms. These terms are merely used to distinguish between information of the same type. For example, without departing from the scope of the one or more embodiments of the present disclosure, the “first” may alternatively be referred to as the “second”. Similarly, the “second” may alternatively be referred to as the “first”. Depending on the context, for example, a word “if” used herein may be interpreted as “while”, “when,” or “in response to determining”.

In addition, it should be noted that user information (including, but not limited to, user equipment information, user personal information, and the like) and data (including, but not limited to, data for analysis, stored data, presented data, and the like) involved in one or more embodiments of the present disclosure are all information and data authorized by a user or sufficiently authorized by all parties. In addition, collection, use, and processing of related data need to comply with related laws, regulations, and standards of related countries and regions, and a corresponding operation entry is provided for a user to select authorization or rejection.

In one or more embodiments of the present disclosure, a large model refers to a deep learning

model having large-scale model parameters, and generally includes hundreds of millions, tens of billions, hundreds of billions, trillions, or even more than ten trillion model parameters. The large model may also be referred to as a cornerstone model/foundation model (Foundation Model). The large model is pre-trained by using large-scale unlabeled corpora, to obtain a pre-trained model with more than a hundred million parameters. Such a model can adapt to a wide range of downstream tasks, and has a good generalization capability, for example, a large language model (Large Language Model, LLM) or a multi-modal pre-training model (multi-modal pre-training model).

During actual application, the large model may be applied to different tasks by using only a small quantity of samples to perform fine-tuning on the pre-training model. The large model may be widely applied to fields such as natural language processing (Natural Language Processing, NLP for short) and computer vision, and specifically may be applied to computer vision field tasks such as visual question answering (Visual Question Answering, VQA for short), image caption (Image Caption, IC), and image generation, and natural language processing field tasks such as text-based sentiment classification, text abstract generation, and machine translation. Main application scenarios of the large model include a digital assistant, an intelligent robot, search, online education, office software, e-commerce, intelligent design, and the like.

First, the terms involved in one or more embodiments of the present disclosure are explained.

Intelligent coding assistant: It uses a large model technology to enhance a programming task in a software development process, and improve coding efficiency of a developer. Main capabilities include automatic code completion, code question answering, automatic unit test generation, automatic code optimization, and the like.

Automatic code completion: In a process in which a developer writes code, possible code completion items are predicted and suggested in real time, thereby improving a coding speed and accuracy of the developer.

AST: It is an AbstractSyntaxTree (abstract syntax tree), which is an intermediate representation form of source code, and is used to abstract and parse structures of the source code in a tree-like structure. The abstract syntax tree may be briefly referred to as a syntax tree (Syntax tree), and is an abstract representation of a source code syntax structure. The abstract syntax tree represents a syntax structure of a programming language in a form of a tree, and each node of the tree represents a kind of structure in the source code. Each layer of structure is also referred to as a node (Node). One AST may be formed by a single node or hundreds or even thousands of nodes. Their combination may describe program syntax for static analysis.

Cross-file call dependency: If a class or a function defined in a file B is called in a file A, it indicates that there is a cross-file call dependency from B to A. Parsing of the call dependency

needs to be performed based on AST syntax parsing.

Large code model: It is an artificial intelligence model based on deep learning, and is specifically designed to understand and generate computer program code. By extracting patterns from a large amount of source code data through learning, this type of model has functions of automatically generating a code snippet, performing code completion, correcting a code error, writing a document, constructing algorithm logic, and the like based on a natural language description or other context information related to programming.

Call hallucination of the large code model: It refers to an illusion that a user may develop during actual application. To be specific, the user may consider that the large code model can fully comprehend the underlying execution logic and intent of the code just like a human programmer, and can always generate code meeting all requirements accurately and without error. However, currently, although the large code model is significantly developed, and can generate high-quality code in many scenarios, the large code model still has some limitations. For example, although the model can generate code based on provided context, the model may fail to possess the deep comprehension and grasp of complex execution logic or domain-specific knowledge compared to that of an experienced developer.

Prompt: It refers to an inputted text paragraph or phrase, and is used as a start point or guide for generating a model output. The text may be a question, a text description, a dialog, or any form of text input. The model generates corresponding output text based on context and semantic information provided by the prompt. A main idea of the prompt is to convert, by using a template, different training data into a common form used for model training, to improve training efficiency.

With the continuous development of computer technologies, many organizations such as institutions or enterprises develop software based on the actual requirements, to provide services for themselves or users. In a software development process, a developer needs to manually write software code. In a daily development process, it is very common for a developer to call, in a file A, a class, a function, and a member variable defined in a file B. In some training processes for the large code model provided in the present disclosure, automatic code completion tends to be trained only based on a single file, and information importing of classes, functions, and member variables that are defined across files is ignored, always resulting in a significant call hallucination problem in a trained large code model. For example, the present disclosure provides two solutions. One solution is to extract a single code source file in a training process, and to randomly find an end position of code prefix and a start position of code suffix. Then, all code between the two positions is used as a prediction result, and data of <prefix, suffix, middle> is constructed for training. However, the solution has a significant defect. In the training solution, automatic code completion is trained only based on a single file, and information importing of classes, functions,

and member variables that are defined across files is ignored, always resulting in a significant call hallucination problem in a trained large code model.

The other solution is to concatenate files having a closer call relationship within a same project in the training process, so that a class/function/variable that is defined in the file B and that is called in the file A appears before a sequence (sequence) of the file A.

However, this solution also has a significant defect. Because a quantity of classes/functions/variables defined in each file tends to be extremely large, the call hallucination problem remains severe when the length of the training sequence is limited. In addition, in a prediction process, it is impossible to include, in a form of prompt, all classes/functions/variables imported into the current file (also due to a limitation of the length of the sequence). This results in a significant gap (gap) between training and prediction, causing the hallucination problem to remain severe when the final prediction is performed.

Based on this, in the present disclosure, a code processing method is provided. In addition, the present disclosure also relates to a model training method, a code completion method, a data processing apparatus, a model training apparatus, a code completion apparatus, a computing device, a computer-readable storage medium, and a computer program product. Detailed descriptions are provided one by one in the following embodiments.

Referring to FIG. 1, FIG. 1 shows a schematic diagram of an application of a code processing method according to an embodiment of the present disclosure. It can be learned from FIG. 1 that in the code processing method provided in one or more embodiments of the present disclosure, a user can upload written to-be-completed code data to a server 104 through a code processing interface in a terminal 102. The server 104 inputs the to-be-completed code data into a trained automatic code completion model, and obtains completed code data outputted by the automatic code completion model. The completed code data corresponds to the to-be-completed code data, and can be used to complete the to-be-completed code data. Then, the server 104 sends the completed code data to the terminal 102. The terminal 102 can present the completed code data to the user through the code processing interface, so that the user performs software development by using the completed code data generated by the automatic code completion model. In addition, it should be noted that before the server 104 processes the to-be-completed code data by using the automatic code completion model, model training needs to be performed on the automatic code completion model. A manner of the model training may be: obtaining cross-file call dependencies of classes or functions by performing AST parsing on original code data, to obtain call dependency metadata, and including, in a prompt, the call dependency metadata and cross-file call definition statements contained in code files as training samples, to train the automatic code completion model. In this way, a cross-file call hallucination problem of the automatic code completion model

is eliminated, and software development efficiency can be better improved by using the automatic code completion model.

Referring to FIG. 2, FIG. 2 shows a flowchart of a model training method according to an embodiment of the present disclosure, which specifically includes the following steps.

Step 202: determine a first sample code file and a second sample code file that correspond to a to-be-trained code processing model, where a call relationship exists between the first sample code file and the second sample code file.

The to-be-trained code processing model may be understood as a code processing model on which model training needs to be performed. The code processing model may be understood as a model that can process to-be-processed code data and determine target code data. The code processing method in one or more embodiments of the present disclosure is applied to different scenarios, and the target code data also varies. For example, when the code processing method is applied to a code completion scenario, the code processing model may be a code completion model, the to-be-processed code data may be to-be-completed code data, and the target code data may be completed code data. The to-be-completed code data is inputted into the code completion model, to obtain the completed code data.

When the code processing method is applied to a code prediction scenario, the code processing model may be a code prediction model, the to-be-processed code data may be to-be-predicted code data, and the target code data may be predicted code data. The to-be-predicted code data is inputted into the code prediction model, to obtain the predicted code data. The to-be-predicted code data may be understood as one or more code statements, for example, two code statements, written by a user. The predicted code data may be understood as next one or more code statements corresponding to the to-be-predicted code data. Subsequently, the predicted code data is presented to the user, to recommend the predicted code data to the user. This implements recommendation of a code snippet to the user, improves development efficiency of the user, and avoids a problem of low development efficiency caused by completely manually writing code statements by the user in a development process.

When the code processing method is applied to a code detection scenario, the code processing model may be a code detection model, the to-be-processed code data may be to-be-detected code data, and the target code data may be a code detection result. The to-be-detected code data is inputted into the code detection model, to obtain the code detection result. The to-be-detected code data may be understood as one or more code statements, for example, two code statements, written by the user. The code detection result may be understood as a code statement having an error in the to-be-detected code data. Alternatively, the code detection result may be understood as a repair code statement for performing repair processing on a code statement having an error in the to-be-

detected code data. Based on this, code data detection or code data repair is performed for the user, thereby improving development efficiency of the user, and avoiding a problem of a code statement error caused by manually writing code statements by the user in a development process.

It should be noted that in one or more embodiments provided in the present disclosure, the code processing model provided in the present disclosure may be a large model or a large code model.

The first sample code file may be understood as a source code file used as training data, and the second sample code file may also be understood as a source code file used as training data. A code call statement that calls the second sample code file exists in the first sample code file. In other words, there is a call relationship between the first sample code file and the second sample code file.

In one or more embodiments provided in the present disclosure, before the to-be-trained code processing model is trained, training data (namely, the first sample code file and the second sample code file) needs to be collected. Specifically, two manners may be used. One manner is to select the first sample code file and the second sample code file that are used as samples from code files written by the user. In a process of collecting a large amount of source code data as the training data, collected data (namely, the first sample code file and the second sample code file) relates to various programming languages and fields. In this way, the code processing model can learn from rich source code data, so that data seen in the training process is close to a real development scenario of the user, and knowledge density and a generalization capability of the code processing model are improved, thereby providing a valuable completion suggestion for any developer in any development scenario.

The other manner is to obtain the first sample code file and the second sample code file from a preset training data set. For example, one option is based on a The-Stack data set. A large quantity of source data files including source code data of projects in various languages, totaling about 3T of data, are acquired and processed in the data set.

In one or more embodiments provided in the present disclosure, the determining a first sample code file and a second sample code file that correspond to a to-be-trained code processing model includes:

determining a plurality of sample code files corresponding to the to-be-trained code processing model, and performing syntax analysis on each sample code file, to obtain code call data in each sample code file; and

determining, based on the code call data, the first sample code file and the second sample code file that has the call relationship with the first sample code file from the sample code files.

The sample code files may be understood as source code files used as samples, and are used

to train the to-be-trained code processing model. The first sample code file and the second sample code file are determined from the plurality of sample code files.

Performing syntax analysis on each sample code file may be understood as performing syntax parsing on source code data in each sample code file, to obtain the code call data in each sample code file. The code call data may be understood as meta-information that will be called across files, such as a class defined in each sample code file, a function included inside or outside a class, a member variable included in a class, etc.

The call relationship may be understood as a cross-file call dependency relationship. The call dependency relationship may be understood as a call relationship or a dependency relationship, and is used to represent a caller-callee relationship between files. In one or more embodiments provided in the present disclosure, the first sample code file may be a call sample code file that initiates a call in the call dependency relationship, and the second sample code file may be a called sample code file that is called by the first sample code file (namely, the call sample code file) in the call dependency relationship.

The model training method is described by using an example in which the model training method provided in the present disclosure is applied to resolving a cross-file call hallucination scenario. In the model training method provided in the present disclosure, during a model training operation, syntax parsing needs to be performed on the source code data, to obtain the cross-file dependency relationship. Specifically, AST syntax parsing is performed on the source code, to obtain the meta-information that will be called across files, such as a class defined in each file, a function included inside or outside a class, a member variable included in a class, etc.

After syntax parsing is completed, the cross-file dependency relationship needs to be obtained through parsing. Referring to FIG. 3, FIG. 3 is a schematic diagram of a sample code file of a model training method according to an embodiment of the present disclosure. A current file in FIG. 3 may be understood as a first sample code file, and a file (file_reader) that the current file depends on for calling and a file (sorter) that the current file depends on for calling may be understood as second sample code files. In other words, there may be one or more first sample code files, and each first sample code file may have one or more corresponding second sample code files. It can be learned from FIG. 3 that, to obtain the cross-file dependency relationship through parsing, a call statement, in source code of the current file, for calling a class and function in the file “sorter” or the file “file_reader” needs to be parsed. In the current file, “data=file_reader.read_json(data_path)” is a class call statement for the file “file_reader” , and “Sorter=sorter()” is a class call statement for the file “sorter”. A call dependency relationship between files is established based on call relationships of classes and functions between the files.

In addition, a process of obtaining the cross-file dependency relationship through parsing may

be implemented by using a dependency relationship analysis tool. For example, the dependency relationship analysis tool may be based on a tree-sitter tool. The tree-sitter tool can analyze the dependency relationship.

Step 204: determine first sample code data and a sample code label based on the first sample code file.

The first sample code data may be understood as one or more code statements that are obtained from the first sample code file and that are used as training samples. In one or more embodiments provided in the present disclosure, the first sample code data and the sample code label may be extracted from the first sample code data. Through this step, model training is performed by using the sample code data and the sample code label that are highly correlated, thereby improving performance of code processing performed by the code processing model. In addition, the to-be-processed code data is efficiently processed by using the code processing model having high performance, thereby improving software development efficiency, and avoiding a problem caused by low software development efficiency. In addition, through the step of "extracting the first sample code data and the sample code label from the first sample code data", a problem of poor performance of a trained model caused by using a technical solution with separated samples and labels is avoided. The technical solution with separated samples and labels may be understood as a technical solution of using source code files as samples and adding corresponding labels to the samples. Because the samples and the labels in this solution are independent of each other and not highly correlated, it results in that the trained model has poor performance.

The sample code label may be understood as one or more code call statements that are obtained from the first sample code file and that are used as sample labels. The code call statement may be understood as a call statement in the first sample code file that can call the second sample code file.

In one or more embodiments provided in the present disclosure, the determining first sample code data and a sample code label based on the first sample code file includes:

determining, from the first sample code file, a code call statement that calls the second sample code file, and determining the code call statement as the sample code label; and

determining, from the first sample code file, other code statements apart from the sample code label, and determining the other code statements as the first sample code data.

The code call statement may be understood as a code statement, in the first sample code file, that relates to cross-file calling of the second sample code file. For example, in "data=file_reader.read_json(data_path)", file_reader may be a called source code file (namely, the second sample code file).

The foregoing example is still used. The code processing model may be an automatic code completion and repair model. Based on this, to construct training data, it can be learned from FIG. 3 that, in this model training method, a statement "data=file_reader.read_json(data_path)" related to cross-file calling in the current file (the first sample code file) is sampled first, and this is a call statement. Second, after the call statement is determined, the call statement may be used as a sample code label.

Specifically, in a process of parsing the call statement in the current file, the call statement is divided into two types: a class call statement and a function call statement. The class call statement is mainly a definition part of a class. The function call statement is mainly to call some functions in other files to obtain a desired output result. An objective of the function call statement is to obtain content from other files. For example, an objective of some call statements is to obtain sorting results of bubble sorting and quick sorting. These call statements relate to cross-file calling of classes and functions, and may be used as targets (namely, sample code labels) of the model training process.

Then, code content preceding the call statement in the current file is used as code prefix, and code content following the call statement is used as code suffix. The code prefix and the code suffix are used as training samples (namely, the first sample code data). After the first sample code data and the sample code label are obtained to perform model training, a prediction target of the model is to predict a complete call statement.

Therefore, by obtaining the first sample code data and the sample code label, model training is performed by using the sample code data and the sample code label that are highly correlated, thereby improving the performance of code processing performed by the code processing model. In addition, the to-be-processed code data is efficiently processed by using the code processing model having high performance, thereby improving the software development efficiency, and avoiding the problem caused by the low software development efficiency.

Step 206: determine second sample code data based on the second sample code file, and determine sample code data based on the first sample code data and the second sample code data.

The second sample code data may be understood as code call data used as a training sample, and the code call data may be understood as data/variables in the source code file that are called by other source code files. For example, the code call data may be data that will be called across files, such as a class defined in the source code file, a function included in a class, a function included outside a class, or a member variable included in a class, etc.

The sample code data may be understood as code data used as a training sample, which is used to train the code processing model. The sample code data includes the first sample code data and the second sample code data.

In one or more embodiments provided in the present disclosure, the determining second sample code data based on the second sample code file includes:

determining initial code call data from the second sample code file; and

performing data filtering processing on the initial code call data to obtain target code call data, and using the target code call data as the second sample code data.

Specifically, according to the model training method provided in one or more embodiments of the present disclosure, in a process of constructing model training data, there are two phases of processing steps for the second sample code data. First, the initial code call data is obtained from the second sample code file. Then, data filtering processing is performed on the initial code call data, to obtain the target code call data obtained after data filtering, and the target code call data is used as the second sample code data. In this way, model training is performed by using the code call data obtained after data filtering, thereby further improving code processing performance. Based on this, the second sample code data is obtained from the second sample code file.

In one or more embodiments provided in the present disclosure, the determining initial code call data from the second sample code file includes:

determining code data in the second sample code file, and performing syntax analysis on the code data, to obtain a plurality of pieces of code call data in the code data; and

performing data concatenation on the pieces of code call data, to obtain the initial code call data.

The code data may be understood as source code in the second sample code file. The syntax analysis may be understood as performing syntax parsing on the source code data, and the syntax parsing may be implemented based on the AST.

The foregoing example is still used. Herein, “fromfile_readerimportFileReader” and “fromsorterimportSorter” may be respectively imported into the current file. In other words, a call dependency from file_reader.py to main.py and a call dependency from sorter.py to main.py exist. Herein, file_reader.py and sorter.py may be understood as second sample code files. Therefore, information such as classes/functions/member variables defined in file_reader.py and sorter.py needs to be separately parsed.

Based on this, two files, namely, file_reader.py and sorter.py, are parsed by using the AST, to obtain metadata obtained by parsing the call dependencies, as shown by the call data in FIG. 3. The metadata obtained through parsing includes a FileReader class in file_reader.py. This class includes a plurality of functions, and definition statements of the plurality of functions are obtained through parsing. The metadata further includes a Sorter class in sorter.py. This class includes a plurality of functions, and definition statements of the plurality of functions are obtained through parsing.

In addition, the call data in FIG. 3 may be understood as second sample code data obtained by parsing the second sample code file.

Finally, all data obtained through parsing is concatenated together, to finally obtain metadata of all content that the current file depends on. In other words, call dependency metadata (namely, the target code call data) includes all information of classes/functions/member variables in other files that the current file depends on, and the information is a basis for resolving call hallucination.

In one or more embodiments provided in the present disclosure, when call dependency metadata <import> (namely, the target code call data) is constructed, the following problems require attention: 1. In the current file, there may be an excessively large quantity of imported statements, but it is possible that not all imported information is actually called in the source code. 2. In a dependent file (the second sample code file), excessively large quantities of classes and functions may be defined, but most of these classes and functions are not actually called.

Due to these two aspects of problems, an excessively large amount of call dependency metadata content may be finally constructed. This not only introduces significant interference to model prediction, but also leads to a sharp expansion in the context length, slowing down an inference speed. To resolve the aspects of problems, the model training method provided in one or more embodiments of the present disclosure uses a negative sampling manner, to ensure that definition information corresponding to actually called statements appears in the dependency metadata, without causing an excessively long context length of the dependency metadata. A specific process is as follows.

The performing data filtering processing on the initial code call data to obtain target code call data includes:

determining first code call data, second code call data, and third code call data from the initial code call data, where the first code call data is data called by the first sample code file, the second code call data is data imported into the second sample code file, and the third code call data is globally defined data;

determining a first data amount of the first code call data;

selecting fourth code call data corresponding to the first code call data from the second code call data and the third code call data based on the first data amount and a preset amount ratio; and

determining the first code call data and the fourth code call data as the target code call data.

The first code call data may be understood as a class/function/member variable that is definitely called, among all call statements of the source code file. The second code call data may be understood as a class/function/member variable that is not called but is imported, among all imported statements of the source code file. The third code call data may be understood as a class/function/member variable that is not called or imported but is defined, among all imported

statements of the source code file.

The first data amount may be understood as a quantity of pieces of data corresponding to the first code call data. When the first code call data is the call dependency metadata, the first data amount may be a quantity of pieces of metadata of the call dependency metadata. The preset amount ratio may be set based on an actual application scenario. This is not specifically limited in the present disclosure.

The foregoing example is still used. A manner of performing negative sampling on the call dependency metadata is: parsing all defined classes/functions/member variables within a project corresponding to the current file (namely, the second sample call file); parsing all imported statements in the current file, to obtain classes/functions/member variables that may be imported; parsing all call statements in the current file, to obtain classes/functions/member variables that are definitely called, using the same as positive data, and storing the positive data into the call dependency metadata; sampling some classes/functions/member variables that are not called but are imported, using the same as negative data, and storing the negative data into the call dependency metadata; and sampling some classes/functions/member variables that are not imported but are defined, using the same as negative data, and storing the negative data into the call dependency metadata. It should be noted that in a process of optimizing the call dependency metadata through negative sampling, a ratio (namely, the preset amount ratio) of the positive data to the negative data is controlled to be approximately 1:5.

The call dependency metadata finally obtained through the foregoing steps includes not only the classes/functions/member variables that are definitely called but also some classes/functions/member variables that are imported but not called or defined. In this way, when the positive data and the negative data both exist, it can help the code processing model to learn of and refer to information about the call dependency metadata, to assist in writing subsequent call statements, and eliminate the call hallucination problem as much as possible.

Step 208: perform model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model.

In one or more embodiments provided in the present disclosure, the performing model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model includes:

inputting the sample code data into the to-be-trained code processing model, to obtain target code data corresponding to the sample code data;

determining a loss value based on the target code data corresponding to the sample code data and the sample code label; and

performing model training on the to-be-trained code processing model based on the loss value

until a model training stop condition is met, to obtain the trained code processing model.

The foregoing example is still used. The automatic code completion model is trained by using the training sample. A training manner is: first, inputting the training sample (namely, the sample code data) into the automatic code completion model, to obtain a prediction call statement (namely, the target code data corresponding to the sample code data) outputted by the model; and next, calculating the loss value by using the prediction call statement and a sample label (namely, the sample code label). Specifically, according to the model training method in one or more embodiments of the present disclosure, the loss value may be calculated by using a loss function based on an actual application scenario. The loss function may be a mean square error loss function. The loss function may be used to calculate an average square distance between a model prediction value (namely, the prediction call statement) and a true target value (namely, the sample label), and the average square distance is used as the loss value. It should be noted that in one or more embodiments provided in the present disclosure, the loss value may be calculated through steps such as forward propagation, loss calculation, averaging, back propagation, and optimization.

Finally, model parameter adjustment is performed on the automatic code completion model based on the loss value, until a model training stop condition is met, to obtain a trained automatic code completion model. The model training stopping condition may be set based on an actual scenario, for example, the model training reaches a preset quantity of rounds, or the loss value converges.

Based on this, when the trained code processing model is obtained, the trained code processing model can be subsequently used to process the to-be-processed code data in the software development process, thereby improving software development efficiency, and avoiding a problem of a large waste of time costs and labor costs in the software development process.

In one or more embodiments provided in the present disclosure, the performing model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model includes:

processing the sample code data and the sample code label by using a training data template, to obtain model training data; and

performing model training on the to-be-trained code processing model based on the model training data, to obtain the trained code processing model.

The training data template may be understood as a template for converting different training data into a common form used for model training. The training data template may be a prompt.

The foregoing example is still used. After the sample code data and the sample code label are obtained, the training data needs to be constructed based on the sample code data and the sample code label. Specifically, for a code block obtained by sampling a large quantity of cross-file call

statements, the preceding code (namely, code prefix) and the following code (namely, code suffix) of the code block are used as context. Its objective is that the model correctly writes a call statement, and content including a definition part of the call statement needs to be included in the prompt, to eliminate the call hallucination problem in the model training process. In other words, the call dependency metadata obtained through parsing in the foregoing embodiments is also included in the prompt as additional information. For example, a prompt (training data) may be finally organized as follows: <filename> current file name <import> call dependency metadata <fim_prefix> code prefix <fim_suffix> code suffix <fim_middle> target call statement, which may be specifically found in FIG. 4. FIG. 4 is a schematic diagram of a prompt of a model training method according to an embodiment of the present disclosure.

In one or more embodiments of the present disclosure, the model training method further includes:

determining to-be-processed code data; and

inputting the to-be-processed code data into the trained code processing model, and performing a data adjustment operation on the to-be-processed code data by using the trained code processing model, to obtain target code data corresponding to the to-be-processed code data, where the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation.

The to-be-processed code data may be understood as code data that needs to be processed by using the code processing model, and the to-be-processed code data may be one or more code statements, for example, “data_p**th=’ data.j**on’ ”. Alternatively, the to-be-processed code data may be one or more code files. In one or more embodiments provided in the present disclosure, the to-be-processed code data may be to-be-completed code data, and the to-be-completed code data may be understood as one or more to-be-completed code statements not written completely, or a to-be-completed code file not written completely. For example, some content not written completely exists in a to-be-completed code statement “data_p**th=”.

The code completion operation may be understood as an operation of performing code completion on to-be-completed code data not completely written.

The code generation operation may be understood as an operation of generating corresponding subsequent code data for the written part of code. The code generation operation may generate corresponding subsequent code based on a part of code in a code file not completely written, to obtain a complete code file that is completely written.

The code repair operation may be understood as an operation of repairing code data that needs to be repaired in the software development process. For example, for code data that is completely

written but has a defect in a code statement, the code repair operation may be performed on the code data for repair, to obtain code data that is completely written and has no defect in the code statement.

In one or more embodiments provided in the present disclosure, the determining to-be-processed code data includes:

obtaining to-be-processed code data written by a user by using a client.

The user may encode the to-be-processed code data by using a code writing system, and in a writing process, the to-be-processed code data written by the user may be obtained.

The code writing system may be understood as an intelligent coding assistant. The intelligent coding assistant can implement functions such as automatic code completion, code snippet recommendation, and error detection and repair based on an advanced artificial intelligence technology and by deeply understanding a programming intention of a developer and context information, thereby significantly improving coding efficiency of the developer. The code processing model provided in one or more of the present disclosure may be deployed in the code writing system, to provide assistance for development work of the developer, thereby improving development efficiency.

Specifically, obtaining the to-be-processed code data written by the user may be understood as detecting the to-be-processed code data written by the user, to obtain the to-be-processed code data. Alternatively, after the user writes the to-be-processed code data, the to-be-processed code data may be provided to a server through a task processing interface in the client, so that the server obtains the to-be-processed code data. Therefore, it is convenient to subsequently process the to-be-processed code data by using the code processing model, thereby improving software development efficiency. The server may be understood as a server to which the code processing method in one or more embodiments of the present disclosure is applied.

In one or more embodiments provided in the present disclosure, after the trained code processing model is obtained by using the sample code data and the sample code label, the to-be-processed code data is inputted into the code processing model, to obtain the target code data corresponding to the to-be-processed code data.

In one or more embodiments provided in the present disclosure, after inputting the to-be-processed code data into the code processing model, to obtain the target code data corresponding to the to-be-processed code data, the method further includes:

sending the target code data to the client, such that the client provides the target code data to the user.

Specifically, after the to-be-processed code data written by the user is inputted into the code processing model, to obtain the target code data corresponding to the to-be-processed code data,

the target code data may be sent to the client, such that the client provides the target code data to the user, thereby facilitating efficient software development work performed by the user based on the target code data, and improving software development efficiency.

According to the model training method in one or more embodiments of the present disclosure, the first sample code file and the second sample code file that has a call relationship with the first sample code file may be determined. In addition, model training is performed based on the first sample code data and the sample code label that are determined based on the first sample code file, and the second sample code data determined based on the second sample code file. In this way, in the model training process, the code processing model is trained by using the sample code files that have the call relationship with each other, so that the trained code processing model can process code data having a relatively complex call relationship in an actual development scenario, thereby improving performance of the code processing model in the actual scenario. In addition, when the code processing model having relatively good performance is applied to the actual development scenario, efficiency of software development can be improved, and software development can be efficiently completed, thereby avoiding a problem that developed software cannot be used in time to provide services.

With reference to FIG. 5, the model training method provided in the present disclosure is further described below by using an example in which the model training method is applied to a cross-file call hallucination scenario of an automatic code completion and repair model. FIG. 5 shows a flowchart of a processing process of a model training method according to an embodiment of the present disclosure, which specifically includes the following steps.

Step 502: collect source code data.

Specifically, in this step, a large amount of source code data is collected for model training. The source code data is collected in two manners.

One manner is: for developers of various fields and various program products, collecting source code data written by the developers in daily work, so that training data in a training process is close to a real development scenario of a user.

The other manner is: determining source code data from a The-Stack data set, where the data set mainly crawls and processes massive warehouses in Github, which includes source code data of projects in various languages, totaling about 3T of data.

Step 504: obtain a cross-file dependency relationship through parsing.

First, AST syntax parsing is performed on the received source code data, to obtain coded data that will be called across files, such as a class defined in a current file (which refers to a written source code data file), a function included inside or outside a class, a member variable included in a class, etc.

Second, all data obtained by parsing the file is concatenated together, to finally obtain metadata of all content that the file depends on.

It should be noted that the foregoing operations are performed on each file, to determine metadata (namely, call data or call dependency metadata) of each file.

Finally, the cross-file dependency relationship is obtained through parsing.

Specifically, obtaining the dependency relationship through parsing may be implemented in two manners.

One manner is:

first, determining, based on the metadata obtained through parsing in step 504, a class and a function in a file B that are called by a file A; and

second, determining, based on the class and the function that are called between the file A and the file B, to establish a call dependency relationship from the file B to the file A.

The other manner is: determining a call dependency relationship between the file A and the file B based on a tree-sitter tool.

The tree-sitter is a parser generation tool and an incremental parsing library, and can parse a code source file into a specific syntax tree, to parse the vast majority of languages.

Step 506: sample a call statement to construct training data.

A specific execution manner of this step is as follows.

1. Determine to sample a statement related to cross-file calling in the current file, and use the statement as a target call statement.

2. Use content preceding the target call statement in the current file as code prefix, and use content following the target call statement as code suffix.

3. Optimize the metadata by using a negative sampling method, to obtain optimized metadata.

Due to some problems, an excessively large amount of call dependency metadata content may be constructed. This not only introduces significant interference to model prediction, but also leads to a sharp expansion in the context length, slowing down an inference speed.

Based on this, the negative sampling method is used to optimize the metadata. A specific implementation manner includes the following.

① Parse all defined classes/functions/member variables within a project corresponding to the current file.

② Parse all imported statements in the current file, to obtain classes/functions/member variables that may be imported.

③ Parse all call statements in the current file, to obtain classes/functions/member variables that are definitely called, use the same as positive data, and store the positive data into the call

dependency metadata.

④ Sample some classes/functions/member variables that are not called but are imported, use the same as negative data, and store the negative data into the call dependency metadata.

⑤ Sample some classes/functions/member variables that are not imported but are defined, use the same as negative data, and store the negative data into the call dependency metadata.

It should be noted that in an optimization process, a ratio of the positive data to the negative data is controlled to be approximately 1:5.

4. Use the target call statement as a sample label, and use the code prefix, the code suffix, and the optimized metadata as training samples, to complete construction of the training data.

5. Include the training samples and the label in a prompt, to construct a piece of training data.

Step 508: train a model.

A specific execution manner of this step is:

first, inputting the training samples into an automatic code completion model, to obtain a prediction call statement outputted by the model;

second, calculating a loss value by using the prediction call statement and the sample label (namely, the target call statement); and

finally, performing model parameter adjustment on the automatic code completion model based on the loss value, until a model training stop condition is met, to obtain a trained automatic code completion model.

In addition, it should be noted that after training is completed, the automatic code completion model may be applied to a code completion scenario, and specific application steps include:

first, obtaining to-be-completed code data written by a programmer;

second, inputting the to-be-completed code data into the trained automatic code completion model, to obtain prediction code (namely, completed code data) outputted by the automatic code completion model, where the prediction code corresponds to the to-be-completed code data, and can complete the to-be-completed code data; and

finally, presenting the prediction code to the user, such that the user uses the prediction code generated by the automatic code completion model.

It can be learned from the foregoing steps that according to the model training method provided in one or more embodiments of the present disclosure, a method for repairing cross-file call hallucination of an automatic code completion model is proposed. The cross-file call dependencies of classes or functions are obtained by performing AST parsing on the original code data, to obtain the call dependency metadata. In addition, the cross-file call definition statements included in the current file and the call dependency metadata are included in the prompt for training,

to eliminate the cross-file call hallucination problem of the model.

In addition, according to the model training method provided in one or more embodiments of the present disclosure, it is proposed to obtain call dependency metadata of cross-file calling through AST syntax parsing. The metadata (namely, the call dependency metadata) integrates information of classes, functions, and member variables across files, to mitigate the call hallucination problem.

According to the model training method provided in one or more embodiments of the present disclosure, it is proposed to summarize information about cross-file calling into call dependency metadata including a definition statement, and training is performed by using a negative sampling technology, thereby greatly reducing the context length, and enabling the training and prediction to have no gap. Therefore, the call hallucination problem can be mitigated and resolved.

In addition, according to the model training method provided in one or more embodiments of the present disclosure, a solution for obtaining call dependency metadata is proposed. Information about the call dependency metadata is integrated by using the negative sampling technology, and some classes and functions that are not called are added, thereby helping the model learn to correctly call the classes and functions defined in the metadata.

Referring to FIG. 6, FIG. 6 shows a flowchart of a code processing method according to an embodiment of the present disclosure, which specifically includes the following steps.

Step 602: receive to-be-processed code data sent by a user through a code processing interface in the client.

Step 604: input the to-be-processed code data into a code processing model, and perform a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data, where the code processing model is obtained through training according to the foregoing model training method, and the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation.

Step 606: send the target code data to the client, such that the client presents the target code data to the user through the code processing interface.

The code processing interface may be understood as an interface presented in the client, and is used for a user to process code. The code processing interface may be an application interface, an internet page, or the like. The code processing interface includes a plurality of types of code processing controls (for example, a button and a text box). The user may process the to-be-processed code data by using the code processing controls. For example, the to-be-processed code data is inputted into a text box, and by clicking a code data processing button, the to-be-processed code data is provided to a server to which the code processing method is applied. In this way, the

to-be-processed code is processed by using the code processing method.

In one or more embodiments provided in the present disclosure, the to-be-processed code data is to-be-completed code data.

The inputting the to-be-processed code data into a code processing model, and performing a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data includes:

inputting the to-be-completed code data into the code processing model, and performing the code completion operation on the to-be-completed code data by using the code processing model, to obtain completed code data corresponding to the to-be-completed code data.

The sending the target code data to the client, such that the client presents the target code data to the user through the code processing interface includes:

sending the completed code data to the client, such that the client presents the completed code data to the user through the code processing interface.

The to-be-completed code data may be understood as one or more to-be-completed code statements not written completely, or a to-be-completed code file not written completely. For example, a code statement "data_p**th= " is a code statement not written completely. After the d code completion operation is performed on the to-be-completed code data by using the code processing model, the corresponding completed code data may be obtained. For example, a code statement "data_p**th=' data.j**on' " may be completed code data corresponding to the foregoing to-be-completed code data.

In one or more embodiments provided in the present disclosure, the to-be-processed code data is to-be-repaired code data.

The inputting the to-be-processed code data into a code processing model, and performing a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data includes:

inputting the to-be-repaired code data into the code processing model, and performing the code repair operation on the to-be-repaired code data by using the code processing model, to obtain repaired code data corresponding to the to-be-repaired code data.

The sending the target code data to the client, such that the client presents the target code data to the user through the code processing interface includes:

sending the repaired code data to the client, such that the client presents the repaired code data to the user through the code processing interface.

The to-be-repaired code data may be understood as one or more to-be-repaired code statements that have a defect in a code writing process, or a to-be-repaired code file that has a

defect in the code writing process. The repaired code data may be understood as one or more completely repaired code statements or a completely repaired code file that are obtained by performing the code repair operation on the to-be-repaired code data by using the code processing model. In the code writing process, a defect such as misspelling or a code logic error may occur on some statements. For these defects, to-be-repaired code data having the defect may be inputted into the code processing model, to obtain repaired code data. For example, a code statement "data_=’ data.j**on’ " has a misspelling problem, where "data_p**th" in the code statement lacks "p**th". The code statement is inputted into the code processing model to perform the code repair operation, to obtain a complete and correct code statement "data_p**th=’ data.j**on’ ".

It should be noted that the client may be understood as a terminal.

According to the code processing method provided in one or more embodiments of the present disclosure, after the to-be-processed code data sent by the user through the code processing interface in the client is received, a data adjustment operation such as a code completion operation, a code generation operation, or a code repair operation can be performed on the to-be-processed code data by using the code processing model, to obtain the target code data. This provides assistance for code writing work of the user in a software development process, improves efficiency of software development, and saves time costs and labor costs in the software development process. In addition, the to-be-processed code data is processed by using the code processing model, thereby avoiding a problem that in a process in which a developer uses a computer device to develop software, a large quantity of computing resources of the computer device are wasted due to low software development efficiency.

It should be noted that the technical solutions of the code processing method and the technical solutions of the foregoing model training method belong to a same concept. For details not described in detail in the technical solutions of the code processing method, reference may be made to the descriptions of the technical solutions of the foregoing model training method.

Referring to FIG. 7, FIG. 7 shows a schematic structural diagram of a code processing platform according to an embodiment of the present disclosure. The code processing platform 700 includes a request receiving unit 702, a code processing unit 704, and a response unit 706, where

the request receiving unit 702 is configured to receive a code processing request, and determine, based on the code processing request, to-be-processed code data and a code processing model identifier corresponding to the to-be-processed code data;

the code processing unit 704 is configured to determine a code processing model based on the code processing model identifier, input the to-be-processed code data into the code processing model, and perform a data adjustment operation on the to-be-processed code data by using the

code processing model, to obtain target code data corresponding to the to-be-processed code data, where the code processing model is obtained through training according to the foregoing model training method, and the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation; and

the response unit 706 is configured to output the target code data.

The code processing platform may be understood as a platform that processes the to-be-processed code data. The code processing platform provides related services for code writing in a software development process. A developer may write code on the code processing platform. In one or more embodiments provided in the present disclosure, the code processing platform may be understood as an intelligent coding assistant. The intelligent coding assistant can implement functions such as automatic code completion, code snippet recommendation, and error detection and repair based on an advanced artificial intelligence technology and by deep understanding a programming intention of a developer and context information, thereby significantly improving coding efficiency of the developer.

The request receiving unit 702 may be understood as a module on the code processing platform that is configured to receive the code processing request. On the code processing platform provided in one or more embodiments of the present disclosure, when a user sends a code processing request to the code processing platform through a code processing interface in a client, the request receiving unit 702 on the code processing platform receives the code processing request, and may also determine to-be-processed code data and a code processing model identifier from the code processing request.

The code processing model identifier may be information uniquely identifying the code processing model. For example, the code processing model identifier may be information such as a code processing model name or a code processing model number, or the code processing model identifier may be a code processing model address corresponding to the code processing model.

The code processing unit 704 may be understood as a unit on the code processing platform that is configured to perform the code adjustment operation, such as the code completion operation, the code generation operation, or the code repair operation, on the to-be-processed code data. The code adjustment operation is implemented by using the code processing model.

The response unit 706 may be understood as a module on the code processing platform that is configured to output the target code data to the outside. After the code processing unit 704 obtains the target code data corresponding to the to-be-processed code data by using the code processing model, the response unit 706 outputs the target code data to the outside.

It should be noted that on the code processing platform provided in one or more embodiments of the present disclosure, the request receiving unit 702, the code processing unit 704, and the

response unit 706 may be hardware modules or software modules.

In one or more embodiments provided in the present disclosure, the request receiving unit 702 is configured to determine the to-be-processed code data and the code processing model identifier corresponding to the to-be-processed code data from the code processing request.

Specifically, in a process in which code processing is performed on the code processing platform, the received code processing request carries the to-be-processed code data and the code processing model identifier corresponding to the to-be-processed code data. In this case, the to-be-processed code data and the code processing model identifier corresponding to the to-be-processed code data may be determined from the code processing request, and then the to-be-processed code data and the code processing model identifier are sent to the code processing unit 704, so that the code processing unit 704 performs the code adjustment operation on the to-be-processed code data based on the code processing model corresponding to the code processing model identifier, to obtain the target code data.

In one or more embodiments provided in the present disclosure, the code processing platform further includes a code question answering function, and the user may input code generation text into the code processing platform through the code processing interface in the client. The code processing platform generates corresponding code data based on the code generation text, thereby meeting an actual application requirement of the user. Specifically, the request receiving unit 702 is configured to determine the code generation text in the code processing request and processing prompt information corresponding to the code generation text, input the code generation text into a code generation model based on the processing prompt information, to obtain the to-be-processed code data, and determine the code processing model identifier corresponding to the to-be-processed code data.

The code generation text may be understood as text used to generate code, and the code generation text may be understood as code description text for code data that needs to be generated. For example, the user may input text data “generate lottery code”, and the code processing platform may generate corresponding lottery code data according to the “generate lottery code”.

The processing prompt information may be understood as prompt information indicating what operations the code processing platform is to perform on the code generation text. The code processing platform provides a plurality of processing functions for code data, to meet requirements of software development work. Based on this, the code processing platform needs to process a code generation file based on the processing prompt information, to avoid a problem that a user requirement cannot be met due to incorrect processing on the code generation file. The processing prompt information may be description information of a processing process of the code generation text. For example, the processing prompt information may be: first, generating the to-

be-processed code data corresponding to the code generation text by using the code generation model, and then performing the code adjustment operation on the to-be-processed code data by using the code processing model corresponding to the code processing model identifier, to obtain the target code data.

The code generation model may be understood as a model that generates corresponding code data based on the code generation text. For example, the text data "generate lottery code" is inputted into the code generation model, and lottery code data corresponding to the "generate lottery code" is generated by using the code generation model.

The code processing platform is described by using an example in which the code processing platform is in a code question answering scenario. The code generation text may be "generate lottery code". Based on this, the user sends the code processing request to the code processing platform through the code processing interface in the client, where the code processing request carries the code generation text and the processing prompt information corresponding to the code generation text. The request receiving unit 702 obtains the code generation text "generate lottery code" and the corresponding processing prompt information from the code processing request after receiving the code processing request. Then, the "generate lottery code" is inputted into the code generation model based on the processing prompt information, to obtain the lottery code data.

After the lottery code data is obtained, to ensure correctness or integrity of the code data, a code processing model identifier of a code processing model that performs a code completion operation or a code repair operation on the lottery code data needs to be determined from the processing prompt information. In addition, the code processing model identifier and the lottery code data are sent to the code processing unit 704.

The code processing unit 704 determines the code processing model corresponding to the code processing model identifier, and performs the code completion operation or the code repair operation on the lottery code data by using the code processing model, to obtain lottery code data having correctness and completeness.

After the lottery code data having correctness and completeness is obtained, the lottery code data is sent to the client through the response module, so that the client presents the lottery code data to the user through the code processing interface. Therefore, during the software development process, the corresponding target code data is generated according to the code generation text, and problems of low code writing efficiency caused by manually writing code by the user and a relatively high error rate of manually writing code are avoided, thereby improving software development efficiency.

According to the code processing platform provided in one or more embodiments of the present disclosure, the request receiving unit can determine, based on the received code processing

request, the to-be-processed code data and the code processing model identifier corresponding to the to-be-processed code data. The code processing unit can perform the data adjustment operation, such as the code completion operation, the code generation operation, or the code repair operation, on the to-be-processed code data by using the code processing model, to obtain the target code data. Finally, the target code data is outputted through the response unit. Therefore, through mutual cooperation between the units, code writing work of the user is assisted in the software development process, thereby improving efficiency of software development and saving time costs and manpower costs in the software development process. In addition, the to-be-processed code data is processed by using the code processing model, thereby avoiding a problem that in a process in which a developer uses a computer device to develop software, a large quantity of computing resources of the computer device are wasted due to low software development efficiency.

It should be noted that the technical solutions of the code processing platform and the technical solutions of the foregoing code processing method or model training method belong to a same concept. For details not described in detail in the technical solutions of the code processing platform, reference may be made to the descriptions of the technical solutions of the foregoing code processing method or model training method.

Corresponding to the foregoing method embodiments, the present disclosure further provides embodiments of a model training apparatus. FIG. 8 shows a schematic structural diagram of a model training apparatus according to an embodiment of the present disclosure. As shown in FIG. 8, the apparatus includes:

a code file determining module 802, configured to determine a first sample code file and a second sample code file that correspond to a to-be-trained code processing model, where a call relationship exists between the first sample code file and the second sample code file;

a first sample determining module 804, configured to determine first sample code data and a sample code label based on the first sample code file;

a second sample determining module 806, configured to determine second sample code data based on the second sample code file, and determine sample code data based on the first sample code data and the second sample code data; and

a model training module 808, configured to perform model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model.

Optionally, the model training apparatus further includes a code processing module, which is configured to:

determine to-be-processed code data; and

input the to-be-processed code data into the trained code processing model, and perform a

data adjustment operation on the to-be-processed code data by using the trained code processing model, to obtain target code data corresponding to the to-be-processed code data, where the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation.

Optionally, the first sample determining module 804 is further configured to:

determine, from the first sample code file, a code call statement that calls the second sample code file, and determine the code call statement as the sample code label; and

determine, from the first sample code file, other code statements apart from the sample code label, and determine the other code statements as the first sample code data.

Optionally, the second sample determining module 806 is further configured to:

determine initial code call data from the second sample code file; and

perform data filtering processing on the initial code call data to obtain target code call data, and use the target code call data as the second sample code data.

Optionally, the second sample determining module 806 is further configured to:

determine code data in the second sample code file, and perform syntax analysis on the code data, to obtain a plurality of pieces of code call data in the code data; and

perform data concatenation on the pieces of code call data, to obtain the initial code call data.

Optionally, the second sample determining module 806 is further configured to:

determine first code call data, second code call data, and third code call data from the initial code call data, where the first code call data is data called by the first sample code file, the second code call data is data imported into the second sample code file, and the third code call data is globally defined data;

determine a first data amount of the first code call data;

select fourth code call data corresponding to the first code call data from the second code call data and the third code call data based on the first data amount and a preset amount ratio; and

determine the first code call data and the fourth code call data as the target code call data.

Optionally, the code file determining module 802 is further configured to:

determine a plurality of sample code files corresponding to the to-be-trained code processing model, and perform syntax analysis on each sample code file, to obtain code call data in each sample code file; and

determine, based on the code call data, the first sample code file and the second sample code file that has the call relationship with the first sample code file from the sample code files.

Optionally, the model training module 808 is further configured to:

process the sample code data and the sample code label by using a training data template, to obtain model training data; and

perform model training on the to-be-trained code processing model based on the model training data, to obtain the trained code processing model.

According to the model training apparatus provided in one or more embodiments of the present disclosure, the first sample code file and the second sample code file that has a call relationship with the first sample code file may be determined, and model training may be performed through the first sample code data and the sample code label that are determined based on the first sample code file, and the second sample code data determined based on the second sample code file. In this way, in the model training process, the code processing model is trained by using the sample code files that have the call relationship with each other, so that the trained code processing model can process code data having a relatively complex call relationship in an actual development scenario, thereby improving performance of the code processing model in the actual scenario. In addition, when the code processing model having relatively good performance is applied to the actual development scenario, efficiency of software development can be improved, and software development can be efficiently completed, thereby avoiding a problem that developed software cannot be used in time to provide services.

The foregoing shows schematic solutions of the model training apparatus according to the embodiments. It should be noted that the technical solutions of the model training apparatus and the technical solutions of the foregoing model training method belong to a same concept. For details not described in detail in the technical solutions of the model training apparatus, reference may be made to the descriptions of the technical solutions of the foregoing model training method.

Corresponding to the foregoing method embodiments, the present disclosure further provides embodiments of a code processing apparatus. FIG. 9 shows a schematic structural diagram of a code processing apparatus according to an embodiment of the present disclosure. As shown in FIG. 9, the apparatus includes:

a code data receiving module 902, configured to receive to-be-processed code data sent by a user through a code processing interface in the client;

a code data processing module 904, configured to input the to-be-processed code data into a code processing model, and perform a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data, where the code processing model is obtained through training according to the foregoing model training method, and the data adjustment operation includes a code completion operation, a code generation operation, or a code repair operation; and

a code data sending module 906, configured to send the target code data to the client, so that the client presents the target code data to the user through the code processing interface.

Optionally, the to-be-processed code data is to-be-completed code data;

the code data processing module 904 is further configured to:

input the to-be-completed code data into the code processing model, and perform the code completion operation on the to-be-completed code data by using the code processing model, to obtain completed code data corresponding to the to-be-completed code data; and

the code data sending module 906 is further configured to:

send the completed code data to the client, so that the client presents the completed code data to the user through the code processing interface.

Optionally, the to-be-processed code data is to-be-repaired code data;

the code data processing module 904 is further configured to:

input the to-be-repaired code data into the code processing model, and perform the code repair operation on the to-be-repaired code data by using the code processing model, to obtain repaired code data corresponding to the to-be-repaired code data; and

the code data sending module 906 is further configured to:

send the repaired code data to the client, so that the client presents the repaired code data to the user through the code processing interface.

According to the code processing apparatus provided in one or more embodiments of the present disclosure, after the to-be-processed code data sent by the user through the code processing interface in the client is received, the data adjustment operation such as the code completion operation, the code generation operation, or the code repair operation can be performed on the to-be-processed code data by using the code processing model, to obtain the target code data, thereby helping code writing work of the user in a software development process, improving efficiency of software development, and saving time costs and manpower costs in the software development process. In addition, the to-be-processed code data is processed by using the code processing model, thereby avoiding a problem that in a process in which a developer uses a computer device to develop software, a large quantity of computing resources of the computer device are wasted due to low software development efficiency.

The foregoing shows schematic solutions of the code processing apparatus according to the embodiments. It should be noted that the technical solutions of the code processing apparatus and the technical solutions of the foregoing code processing method belong to a same concept. For details not described in detail in the technical solutions of the code processing apparatus, reference may be made to the descriptions of the technical solutions of the foregoing code processing method.

FIG. 10 shows a structural block diagram of a computing device 1000 according to an embodiment of the present disclosure. Components of the computing device 1000 include, but are not limited to, a memory 1010 and a processor 1020. The processor 1020 and the memory 1010 are connected through a bus 1030. A database 1050 is used to save data.

The computing device 1000 further includes an access device 1040. The access device 1040 enables the computing device 1000 to communicate via one or more networks 1060. Examples of these networks include a public switched telephone network (PSTN, Public Switched Telephone Network), a local area network (LAN, Local Area Network), a wide area network (WAN, Wide Area Network), a personal area network (PAN, Personal Area Network), or a combination of communication networks, for example, the Internet. The access device 1040 may include one or more of wired or wireless network interfaces (for example, a network interface controller (NIC, network interface controller)) of any type, such as an IEEE802.11 wireless local area network (WLAN, Wireless Local Area Network) wireless interface, a worldwide interoperability for microwave access (Wi-MAX, Worldwide Interoperability for Microwave Access) interface, an Ethernet interface, a universal serial bus (USB, Universal Serial Bus) interface, a cellular network interface, a Bluetooth interface, and near field communication (NFC, Near Field Communication).

In an embodiment of the present disclosure, the foregoing components and other components not shown in FIG. 10 of the computing device 1000 may be connected to each other, for example, connected via the bus. It should be understood that the structural block diagram of the computing device shown in FIG. 10 is merely exemplary, and is not intended to limit the scope of the present disclosure. A person skilled in the art may add, or make replacement with, other components according to requirements.

The computing device 1000 may be a stationary or mobile computing device of any type, including a mobile computer or a mobile computing device (for example, a tablet computer, a person digital assistant, a laptop computer, a notebook computer, or a netbook), a mobile phone (for example, a smartphone), a wearable computing device (for example, a smartwatch or smart glasses), or mobile devices of other types, or stationary computing devices such as a desktop computer or a personal computer (PC, Personal Computer). The computing device 1000 may also be a mobile or stationary server.

The processor 1020 is configured to execute the computer program/computer instructions which, when executed by the processor, implement steps of the foregoing model training method or steps of the foregoing code processing method.

The embodiments in the present disclosure are all described in a progressive manner. For same or similar parts in the embodiments, reference may be made to each other, and descriptions of each embodiment focus on a difference from other embodiments. In particular, the computing device embodiments are basically similar to the model training method embodiments or the code processing method embodiments, and therefore are described briefly. For related parts, reference may be made to partial descriptions of the model training method embodiments or the code processing method embodiments.

According to an embodiment of the present disclosure, a computer-readable storage medium is further provided. The computer-readable storage medium stores a computer program/computer instructions, and when the computer program/computer instructions are executed by a processor, steps of the foregoing model training method or steps of the foregoing code processing method are implemented.

The embodiments in the present disclosure are all described in a progressive manner. For same or similar parts in the embodiments, reference may be made to each other, and descriptions of each embodiment focus on a difference from other embodiments. In particular, the computer-readable storage medium embodiments are basically similar to the model training method embodiments or the code processing method embodiments, and therefore are described briefly. For related parts, reference may be made to partial descriptions of the model training method embodiments or the code processing method embodiments.

According to an embodiment of the present disclosure, a computer program product is further provided. The computer program product includes a computer program/computer instructions, and when the computer program/computer instructions are executed by a processor, steps of the foregoing model training method or steps of the foregoing code processing method are implemented.

The foregoing describes a schematic solution of the computer program product of this embodiment. It should be noted that the technical solutions of the computer program product and the technical solutions of the foregoing model training method or code processing method belong to a same concept. For details not described in detail in the technical solutions of the computer program product, reference may be made to the descriptions of the technical solutions of the foregoing model training method or code processing method.

The foregoing describes specific embodiments of the present disclosure. Other embodiments fall within the scope of the appended claims. In some cases, the actions or steps recited in the claims may be performed in a different order than those in the embodiments and the desired results can still be achieved. In addition, the processes depicted in the accompanying drawings do not necessarily require the particular order shown or sequential order to achieve the desired results. In some embodiments, multitasking and parallel processing are also possible or may be advantageous.

The computer instructions include computer program code. The computer program code may be in a source code form, an object code form, executable file or some intermediate forms, or the like. The computer-readable medium may include: any entity or apparatus that is capable of carrying the computer program code, a recording medium, a USB flash drive, a removable hard disk, a magnetic disk, an optical disc, a computer memory, a read-only memory (ROM, Read-Only Memory), a random access memory (RAM, Random Access Memory), an electric carrier signal,

a telecommunication signal, a software distribution medium, and the like. It should be noted that content included in the computer-readable medium may be properly increased or reduced according to requirements of patent practices. For example, in some regions, according to the patent practices, the computer readable medium does not include an electric carrier signal or a telecommunication signal.

It should be noted that the foregoing method embodiments are each expressed as a combination of a series of actions for the purpose of brief descriptions. However, it should be understood by a person skilled in the art that because some steps may be performed in other sequences or simultaneously according to the embodiments of the present disclosure, the embodiments of the present disclosure are not limited to the described action sequence. In addition, it should be understood by a person skilled in the art that the embodiments described in this specification are all exemplary embodiments, and actions and modules involved are not necessarily mandatory in the embodiments of the present disclosure.

In the foregoing embodiments, the descriptions of the embodiments have respective focuses. For a part that is not described in detail in an embodiment, reference may be made to related descriptions in other embodiments.

The foregoing disclosed preferably embodiments of the present disclosure are only used to help explain the present disclosure. The optional embodiments are not described in full detail, and are not intended to limit the invention to the specific implementations described. Obviously, many modifications and changes may be made according to the content of the embodiments of the present disclosure. These embodiments are selected and described in the present disclosure for better explaining the principles and practical applications of the embodiments of the present disclosure, so that a person skilled in the art can better understand and use the present disclosure. The present disclosure is limited only by the claims and full scope and equivalents thereof.

CLAIMS

1. A model training method, comprising:

determining a first sample code file and a second sample code file that correspond to a to-be-trained code processing model, wherein a call relationship exists between the first sample code file and the second sample code file;

determining first sample code data and a sample code label based on the first sample code file;

determining second sample code data based on the second sample code file, and determining sample code data based on the first sample code data and the second sample code data; and

performing model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model.

2. The model training method according to claim 1, further comprising:

determining to-be-processed code data; and

inputting the to-be-processed code data into the trained code processing model, and performing a data adjustment operation on the to-be-processed code data by using the trained code processing model, to obtain target code data corresponding to the to-be-processed code data, wherein the data adjustment operation comprises a code completion operation, a code generation operation, or a code repair operation.

3. The model training method according to claim 1 or 2, wherein the determining first sample code data and a sample code label based on the first sample code file comprises:

determining, from the first sample code file, a code call statement that calls the second sample code file, and determining the code call statement as the sample code label; and

determining, from the first sample code file, other code statements apart from the sample code label, and determining the other code statements as the first sample code data.

4. The model training method according to any one of claims 1 to 3, wherein the determining second sample code data based on the second sample code file comprises:

determining initial code call data from the second sample code file; and

performing data filtering processing on the initial code call data to obtain target code call data, and using the target code call data as the second sample code data.

5. The model training method according to claim 4, wherein the determining initial code call data from the second sample code file comprises:

determining code data in the second sample code file, and performing syntax analysis on the code data, to obtain a plurality of pieces of code call data in the code data; and

performing data concatenation on the pieces of code call data, to obtain the initial code call data.

6. The model training method according to claim 4 or 5, wherein the performing data filtering processing on the initial code call data to obtain target code call data comprises:

determining first code call data, second code call data, and third code call data from the initial code call data, wherein the first code call data is data called by the first sample code file, the second code call data is data imported into the second sample code file, and the third code call data is globally defined data;

determining a first data amount of the first code call data;

selecting fourth code call data corresponding to the first code call data from the second code call data and the third code call data based on the first data amount and a preset amount ratio; and

determining the first code call data and the fourth code call data as the target code call data.

7. The model training method according to any one of claims 1 to 6, wherein the determining a first sample code file and a second sample code file that correspond to a to-be-trained code processing model comprises:

determining a plurality of sample code files corresponding to the to-be-trained code processing model, and performing syntax analysis on each sample code file, to obtain code call data in each sample code file; and

determining, based on the code call data, the first sample code file and the second sample code file that has the call relationship with the first sample code file from the sample code files.

8. The model training method according to any one of claims 1 to 7, wherein the performing model training on the to-be-trained code processing model based on the sample code data and the sample code label, to obtain a trained code processing model comprises:

processing the sample code data and the sample code label by using a training data template, to obtain model training data; and

performing model training on the to-be-trained code processing model based on the model training data, to obtain the trained code processing model.

9. A code processing method, comprising:

receiving to-be-processed code data sent by a user through a code processing interface in a client;

inputting the to-be-processed code data into a code processing model, and performing a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data, wherein the code processing model is obtained through training according to the model training method according to any one of claims 1 to 8, and the data adjustment operation comprises a code completion operation, a code generation operation, or a code repair operation; and

sending the target code data to the client, such that the client presents the target code data to

the user through the code processing interface.

10. The code processing method according to claim 9, wherein the to-be-processed code data is to-be-completed code data;

the inputting the to-be-processed code data into a code processing model, and performing a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data comprises:

inputting the to-be-completed code data into the code processing model, and performing the code completion operation on the to-be-completed code data by using the code processing model, to obtain completed code data corresponding to the to-be-completed code data; and

the sending the target code data to the client, such that the client presents the target code data to the user through the code processing interface comprises:

sending the completed code data to the client, such that the client presents the completed code data to the user through the code processing interface.

11. The code processing method according to claim 9, wherein the to-be-processed code data is to-be-repaired code data;

the inputting the to-be-processed code data into a code processing model, and performing a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data comprises:

inputting the to-be-repaired code data into the code processing model, and performing the code repair operation on the to-be-repaired code data by using the code processing model, to obtain repaired code data corresponding to the to-be-repaired code data; and

the sending the target code data to the client, such that the client presents the target code data to the user through the code processing interface comprises:

sending the repaired code data to the client, such that the client presents the repaired code data to the user through the code processing interface.

12. A code processing platform, comprising a request receiving unit, a code processing unit, and a response unit, wherein

the request receiving unit is configured to receive a code processing request, and determine, based on the code processing request, to-be-processed code data and a code processing model identifier corresponding to the to-be-processed code data;

the code processing unit is configured to determine a code processing model based on the code processing model identifier, input the to-be-processed code data into the code processing model, and perform a data adjustment operation on the to-be-processed code data by using the code processing model, to obtain target code data corresponding to the to-be-processed code data, wherein the code processing model is obtained through training according to the model training

method according to any one of claims 1 to 8, and the data adjustment operation comprises a code completion operation, a code generation operation, or a code repair operation; and

the response unit is configured to output the target code data.

13. The code processing platform according to claim 12, wherein the request receiving unit is configured to determine, in the code processing request, the to-be-processed code data and the code processing model identifier corresponding to the to-be-processed code data from the code processing request; or

the request receiving unit is configured to determine code generation text and processing prompt information corresponding to the code generation text in the code processing request, input the code generation text into a code generation model based on the processing prompt information, to obtain the to-be-processed code data, and determine the code processing model identifier corresponding to the to-be-processed code data.

14. A computing device, comprising:

a memory and a processor, wherein

the memory is configured to store a computer program/computer instructions, the processor is configured to execute the computer program/computer instructions, and when the computer program/computer instructions are executed by the processor, steps of the model training method according to any one of claims 1 to 8 or steps of the code processing method according to any one of claims 9 to 11 are implemented.

15. A computer-readable storage medium, storing a computer program/computer instructions, wherein when the computer program/computer instructions are executed by a processor, steps of the model training method according to any one of claims 1 to 8 and steps of the code processing method according to any one of claims 9 to 11 are implemented.

16. A computer program product, comprising a computer program/computer instructions, wherein when the computer program/computer instructions are executed by a processor, steps of the model training method according to any one of claims 1 to 8 and steps of the code processing method according to any one of claims 9 to 11 are implemented.