模型训练方法、代码处理方法以及代码处理平台

本公开要求于 2024 年 03 月 29 日提交中国专利局、申请号为 202410384068.8、申请名称为“模型训练方法、代码处理方法以及代码处理平台”的中国专利申请的优先权，其全部内容通过引用结合在本公开中。

技术领域

本公开实施例涉及计算机技术领域，特别涉及一种模型训练方法、一种代码处理方法以及一种代码处理平台。

背景技术

随着计算机技术的不断发展，许多机构或企业等组织会根据实际需要进行软件开发，从而为自身或者用户提供服务。在进行软件开发的过程中，开发者可以利用神经网络模型为软件代码的编写提供帮助。通过将编写的代码数据输入神经网络模型，从而获得代码数据对应的处理结果。

在利用神经网络模型对代码数据进行处理之前，需要利用代码样本数据对该神经网络模型进行模型训练。但是，现有技术在进行模型训练的过程中，往往采用相互独立的代码样本进行模型训练，这样会导致训练完成的神经网络模型的性能较差，无法较好的对实际开发场景中关系复杂的代码数据进行处理，从而导致软件开发的效率较低，无法及时的利用开发的软件提供服务。因此，亟需提供一种性能良好的、能够对关系复杂的代码数据进行处理的模型。

发明内容

有鉴于此，本公开实施例提供了一种模型训练方法。本公开一个或者多个实施例同时涉及一种代码处理方法，一种模型训练装置，一种代码处理装置，一种代码处理平台，一种计算设备，一种计算机可读存储介质以及一种计算机程序产品，以解决现有技术中存在的神经网络模型的性能较差，无法较好的对实际开发场景中关系复杂的代码数据进行处理的技术缺陷。

根据本公开实施例的第一方面，提供了一种模型训练方法，包括：

确定待训练代码处理模型对应的第一样本代码文件以及第二样本代码文件，其中，所述第一样本代码文件与所述第二样本代码文件存在调用关系；

基于所述第一样本代码文件，确定第一样本代码数据以及样本代码标签；

基于所述第二样本代码文件确定第二样本代码数据，并基于所述第一样本代码数据以及所述第二样本代码数据，确定样本代码数据；

基于所述样本代码数据以及所述样本代码标签，对所述待训练代码处理模型进行模型训练，获得训练完成的代码处理模型。

根据本公开实施例的第二方面，提供了一种模型训练装置，包括：

代码文件确定模块，被配置为确定待训练代码处理模型对应的第一样本代码文件以及第二样本代码文件，其中，所述第一样本代码文件与所述第二样本代码文件存在调用关系；

第一样本确定模块，被配置为基于所述第一样本代码文件，确定第一样本代码数据以及样本代码标签；

第二样本确定模块，被配置为基于所述第二样本代码文件确定第二样本代码数据，并基于所述第一样本代码数据以及所述第二样本代码数据，确定样本代码数据；

模型训练模块，被配置为基于所述样本代码数据以及所述样本代码标签，对所述待训

练代码处理模型进行模型训练，获得训练完成的代码处理模型。

根据本公开实施例的第三方面，提供了一种代码处理方法，包括：

接收用户通过所述客户端中的代码处理界面发送的待处理代码数据；

将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根据上述模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作；

将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示。

根据本公开实施例的第四方面，提供了一种代码处理装置，包括：

代码数据接收模块，被配置为接收用户通过所述客户端中的代码处理界面发送的待处理代码数据；

代码数据处理模块，被配置为将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根据上述模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作；

代码数据发送模块，被配置为将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示。

根据本公开实施例的第五方面，提供了一种代码处理平台，包括请求接收单元、代码处理单元和响应单元；

所述请求接收单元，用于接收代码处理请求，并基于所述代码处理请求确定待处理代码数据以及所述待处理代码数据对应的代码处理模型标识；

所述代码处理单元，用于基于所述代码处理模型标识确定代码处理模型，将所述待处理代码数据输入所述代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根据上述模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作；

所述响应单元，用于输出所述目标代码数据。

根据本公开实施例的第六方面，提供了一种计算设备，包括：

存储器和处理器；

所述存储器用于存储计算机程序/指令，所述处理器用于执行所述计算机程序/指令，该计算机程序/指令被处理器执行时实现上述模型训练方法、代码处理方法的步骤。

根据本公开实施例的第七方面，提供了一种计算机可读存储介质，其存储有计算机程序/指令，该计算机程序/指令被处理器执行时实现上述模型训练方法、代码处理方法的步骤。

根据本公开实施例的第八方面，提供了一种计算机程序产品，包括计算机程序/指令，该计算机程序/指令被处理器执行时实现上述模型训练方法、代码处理方法的步骤。

本公开一个或多个实施例中的模型训练方法，可以确定第一样本代码文件以及与第一样本代码文件存在调用关系的第二样本代码文件，并且，通过基于第一样本代码文件确定的第一样本代码数据以及样本代码标签、基于第二样本代码文件确定第二样本代码数据进行模型训练，实现在模型训练过程中，利用彼此之间存在调用关系的样本代码文件对代码处理模型进行训练，从而使得训练后的代码处理模型，能够对实际开发场景中调用关系较为复杂的代码数据进行处理，提高了实际场景中代码处理模型的性能；并且，在将性能较好的代码处理模型应用于实际开发场景中，能够提高软件开发的效率，实现高效的完成软

件开发，从而避免了无法及时的利用开发的软件提供服务的问题。

附图说明

图 1 是本公开一个实施例提供的一种数据处理方法的应用示意图；

图 2 是本公开一个实施例提供的一种模型训练方法的流程图；

图 3 是本公开一个实施例提供的一种模型训练方法的样本代码文件的示意图；

图 4 是本公开一个实施例提供的一种模型训练方法的 prompt 的示意图；

图 5 是本公开一个实施例提供的一种模型训练方法的处理过程流程图；

图 6 是本公开一个实施例提供的一种代码处理方法的流程图；

图 7 是本公开一个实施例提供的一种代码处理平台的结构示意图；

图 8 是本公开一个实施例提供的一种模型训练装置的结构示意图；

图 9 是本公开一个实施例提供的一种代码处理装置的结构示意图；

图 10 是本公开一个实施例提供的一种计算设备的结构框图。

具体实施方式

在下面的描述中阐述了很多具体细节以便于充分理解本公开。但是本公开能够以很多不同于在此描述的其它方式来实施，本领域技术人员可以在不违背本公开内涵的情况下做类似推广，因此本公开不受下面公开的具体实施的限制。

在本公开一个或多个实施例中使用的术语是仅仅出于描述特定实施例的目的，而非旨在限制本公开一个或多个实施例。在本公开一个或多个实施例和所附权利要求书中所使用的单数形式的“一种”、“所述”和“该”也旨在包括多数形式，除非上下文清楚地表示其他含义。还应当理解，本公开一个或多个实施例中使用的术语“和/或”是指并包含一个或多个相关联的列出项目的任何或所有可能组合。

应当理解，尽管在本公开一个或多个实施例中可能采用术语第一、第二等来描述各种信息，但这些信息不应限于这些术语。这些术语仅用来将同一类型的信息彼此区分开。例如，在不脱离本公开一个或多个实施例范围的情况下，第一也可以被称为第二，类似地，第二也可以被称为第一。取决于语境，如在此所使用的词语“如果”可以被解释成为“在……时”或“当……时”或“响应于确定”。

此外，需要说明的是，本公开一个或多个实施例所涉及的用户信息（包括但不限于用户设备信息、用户个人信息等）和数据（包括但不限于用于分析的数据、存储的数据、展示的数据等），均为经用户授权或者经过各方充分授权的信息和数据，并且相关数据的收集、使用和处理需要遵守相关国家和地区的相关法律法规和标准，并提供有相应的操作入口，供用户选择授权或者拒绝。

本公开一个或多个实施例中，大模型是指具有大规模模型参数的深度学习模型，通常包含上亿、上百亿、上千亿、上万亿甚至十万亿以上的模型参数。大模型又可以称为基石模型/基础模型（Foundation Model），通过大规模无标注的语料进行大模型的预训练，产出亿级以上参数的预训练模型，这种模型能适应广泛的下游任务，模型具有较好的泛化能力，例如大规模语言模型(Large Language Model，LLM)、多模态预训练模型(multi-modal pre-training model）等。

大模型在实际应用时，仅需少量样本对预训练模型进行微调即可应用于不同的任务中，大模型可以广泛应用于自然语言处理（Natural Language Processing，简称 NLP）、计算机视觉等领域，具体可以应用于如视觉问答（Visual Question Answering，简称 VQA）、图像描述（Image Caption，简称 IC）、图像生成等计算机视觉领域任务，以及基于文本的情感分类、文本摘要生成、机器翻译等自然语言处理领域任务，大模型主要的应用场景包括数字助理、智能机器人、搜索、在线教育、办公软件、电子商务、智能设计等。

首先，对本公开一个或多个实施例涉及的名词术语进行解释。

智能编码助手：利用大模型技术，增强软件开发流程中编程任务，提升开发者的编码效率，主要的能力包括代码自动补全、代码问答、自动生成单元测试、自动代码优化等。

代码自动补全：在开发者编写代码的过程中，实时预测并建议可能的代码完成项，提高开发者编码速度和准确性。

AST：AbstractSyntaxTree（抽象语法树），它是源代码的一种中间表示形式，用于将源代码的结构以树状结构进行抽象和解析。抽象语法树可以简称为语法树(Syntax tree)，是源代码语法结构的抽象表示。它以树状的形式表现编程语言的语法结构，树上的每个节点都表示源代码中的一种结构。每一层结构也被叫做节点（Node）。一个 AST 可以由单一的节点或是成百上千个节点构成。它们组合在一起可以描述用于静态分析的程序语法。

跨文件调用依赖：在 A 文件中调用了 B 文件定义的类或函数，则说明存在一条从 B 到 A 的跨文件调用依赖。解析调用依赖需要在 AST 语法解析的基础上进行。

代码大模型：是一种基于深度学习的人工智能模型，特别设计用于理解和生成计算机程序代码。这类模型通过学习从大量源代码数据中抽取模式，从而具备根据自然语言描述或其他编程相关的上下文信息自动生成代码片段、完成代码补全、修改代码错误、编写文档、构建算法逻辑等功能。

代码大模型的调用幻觉：是指在实际应用中，用户可能会产生一种错觉，即认为代码大模型能够像人类程序员那样完全理解代码背后的执行逻辑和意图，并且总能准确无误地生成满足所有需求的代码。然而，目前尽管代码大模型已经取得显著进展，在很多场景下可以生成高质量的代码，但它们仍然存在一些局限性：比如，虽然模型可以基于给定的上下文生成代码，但在复杂执行逻辑或特定领域知识方面，可能无法像经验丰富的开发者那样深入理解和把握。

Prompt：指的是一个输入的文本段落或短语，作为生成模型输出的起点或引导。这个文本可以是一个问题、一段文字描述、一段对话或任何形式的文本输入。模型会根据这个 prompt 所提供的上下文和语义信息，生成相应的输出文本。Prompt 的主要思想，是通过模板将不同的训练数据转换为模型训练时常见的形式，提高训练效率。

随着计算机技术的不断发展，许多机构或企业等组织会根据实际需要进行软件开发，从而为自身或者用户提供服务。在进行软件开发的过程中，需要开发者手动进行软件代码的编写。在开发者日常开发过程中，在 A 文件中调用 B 文件定义的类、函数、成员变量是非常普遍的现象。本公开提供的一些针对代码大模型训练过程中，往往只是基于单个文件训练代码自动补全，而忽视了跨文件定义的类、函数和成员变量的信息导入，这使得训练出来的代码大模型往往会出现明显的调用幻觉问题。例如，本公开提供的两种方案，一种方案是在训练过程中取单个代码源文件，随机找出代码上文的结束位置、代码下文的开始位置。然后将两个位置中间的所有代码作为预测结果，构造<prefix,suffix,middle>的数据进行训练。但是，该方案存在较大缺陷，该训练方案只是基于单个文件训练代码自动补全，而忽视了跨文件定义的类、函数和成员变量的信息导入，这使得训练出来的代码大模型往往会出现明显的调用幻觉问题。

另一种方案是在训练过程中，将同一个项目下，调用关系更近的那些文件拼接在一起，使得 A 文件中调用的 B 文件中定义的类/函数/变量会出现在 A 文件的 sequence（序列）之前。

但是，该方案也存在较大缺陷，由于每个文件中往往定义的类/函数/变量的数量非常多，导致训练序列长度有限的情况下，幻觉问题仍然会很严重。并且由于预测过程中，无法将当前文件中导入的所有的类/函数/变量通过 prompt 的形式全部放进来（也是因为序列长度的限制），会导致训练和预测的 gap（差距）非常明显，导致最终预测时幻觉问题仍然很严重。

基于此，在本公开中，提供了一种代码处理方法，本公开同时涉及一种模型训练方法，一种代码补全方法，一种数据处理装置，一种模型训练装置，一种代码补全装置，一种计算设备，一种计算机可读存储介质以及一种计算机程序产品，在下面的实施例中逐一进行详细说明。

参见图1，图1示出了根据本公开一个实施例提供的一种代码处理方法的应用示意图，基于图1可知，本公开一个或多个实施例中提供的代码处理方法中，用户能够通过终端102中的代码处理界面将编写的待补全代码数据上传至服务端104，该服务端104将待补全代码数据输入训练后的代码自动补全模型，获得代码自动补全模型输出的代码补全数据，其中，该代码补全数据与待补全代码数据相对应，能够对待补全代码数据进行补全。然后，该服务端104将代码补全数据发送至终端102。该终端102能够通过代码处理界面将代码补全数据向用户展示，让用户使用代码自动补全模型生成的代码补全数据进行软件开发。并且需要说明的是，在服务端104利用代码自动补全模型对待补全代码数据进行处理之前，需要对代码自动补全模型进行模型训练。该模型训练的方式可以是：通过对原始的代码数据进行AST解析跨文件类或函数的调用依赖，获取调用依赖元数据，并将代码文件中包含的跨文件调用定义语句和调用依赖元数据作为训练样本放入prompt，对代码自动补全模型进行训练，从而消除代码自动补全模型的跨文件调用的幻觉问题，利用代码自动补全模型能够更好的提高软件开发的效率。

参见图2，图2示出了根据本公开一个实施例提供的一种模型训练方法的流程图，具体包括以下步骤。

步骤202：确定待训练代码处理模型对应的第一样本代码文件以及第二样本代码文件，其中，所述第一样本代码文件与所述第二样本代码文件存在调用关系。

其中，待训练代码处理模型可以理解为需要进行模型训练的代码处理模型。该代码处理模型可以理解为能够对待处理代码数据进行处理，并确定目标代码数据的模型。在本公开一个或多个实施例中的代码处理方法，应用于不同的场景，该目标代码数据也不同。例如，在代码处理方法应用于代码补全场景的情况下，该代码处理模型可以是代码补全模型，待处理代码数据可以是待补全代码数据，目标代码数据可以是代码补全数据。通过将待补全代码数据输入代码补全模型，获得代码补全数据。

在代码处理方法应用于代码预测场景的情况下，该代码处理模型可以是代码预测模型，待处理代码数据可以是待预测代码数据，目标代码数据可以是代码预测数据。通过将待预测代码数据输入代码预测模型，获得代码预测数据。其中，待预测代码数据可以理解为用户编写的一条或多条代码语句，例如两条代码语句。该代码预测数据可以理解为与待预测代码数据对应的下一条或多条代码语句。后续通过将代码预测数据展示给用户，从而将代码预测数据推荐给用户，实现对用户进行代码片段推荐，提高用户的开发效率，避免用户在开发过程中由于完全手动编写代码语句所导致的开发效率较低的问题。

在代码处理方法应用于代码检测场景的情况下，该代码处理模型可以是代码检测模型，待处理代码数据可以是待检测代码数据，目标代码数据可以是代码检测结果。通过将待检测代码数据输入代码检测模型，获得代码检测结果。其中，待检测代码数据可以理解为用户编写的一条或多条代码语句，例如两条代码语句。该代码检测结果可以理解为待检测代码数据存在错误的代码语句。或者，该代码检测结果可以理解为对待检测代码数据中存在错误的代码语句进行修复处理的修复代码语句。基于此，实现对用户进行代码数据检测或代码数据修复，提高用户的开发效率，避免用户在开发过程中由于手动编写代码语句所导致的代码语句错误的问题。

需要说明的是，在本公开提供的一个或多个实施例中，本公开提供的代码处理模型可以是大模型或代码大模型。

第一样本代码文件可以理解为作为训练数据的源代码文件，第二样本代码文件也可以理解为作为训练数据的源代码文件。其中，该第一样本代码文件中存在调用第二样本代码

文件的调用代码语句。也即是说，第一样本代码文件与所述第二样本代码文件存在调用关系。

在本公开提供的一个或多个实施例中，在对待训练代码处理模型进行训练之前，需要进行训练数据（即第一样本代码文件以及第二样本代码文件）的采集。具体可以采用两种方式，一种方式是从用户编写的代码文件中，选择作为样本的第一样本代码文件以及第二样本代码文件。在收集大量源代码数据作为训练数据的过程中，收集到的数据（第一样本代码文件以及第二样本代码文件）涉及各种各样的编程语言、领域，这样可以使得代码处理模型在丰富的源代码数据中进行学习，使得训练过程中见到的数据与用户真实开发场景接近，同时提升代码处理模型的知识密度和泛化能力，从而在任何开发者的任何开发场景中都能提供有价值的补全建议。

另一种方式是从预设训练数据集中获取第一样本代码文件以及第二样本代码文件，例如，一种选择是基于 The-Stack 数据集，该数据集采集并处理了大量的源数据文件，其中包含各种语言的项目的源代码数据，共包含 3T 左右的数据。

在本公开提供的一个或多个实施例中，所述确定待训练代码处理模型对应的第一样本代码文件以及第二样本代码文件，包括：

确定待训练代码处理模型对应的多个样本代码文件，并对各样本代码文件进行语法分析，获得各样本代码文件中的调用代码数据；

基于所述调用代码数据，从所述各样本代码文件中确定所述第一样本代码文件以及与所述第一样本代码文件存在调用关系的所述第二样本代码文件。

其中，样本代码文件可以理解为作为样本的源代码文件，用于对待训练代码处理模型进行训练。其中，第一样本代码文件以及第二样本代码文件从所述多个样本代码文件中确定。

其中，对各样本代码文件进行语法分析可以理解为对各样本代码文件中的源代码数据进行语法解析，从而获得各样本代码文件中的调用代码数据。其中，该调用代码数据可以理解为每个样本代码文件中定义的类、类中或类外包含的函数、类中包含的成员变量等会被跨文件调用的这些元信息。

其中，调用关系可以理解为跨文件的调用依赖关系，该调用依赖关系可以理解为调用关系、依赖关系，是指用于表示文件与文件之间调用与被调用的关系。在本公开提供的一个或多个实施例中，第一样本代码文件可以是调用依赖关系中发起调用的调用样本代码文件，该第二样本代码文件可以是调用依赖关系中被第一样本代码文件（即调用样本代码文件）所调用的被调用样本代码文件。

以本公开提供的模型训练方法在解决跨文件调用幻觉场景下的应用为例，对模型训练方法进行说明，本公开提供的模型训练方法在进行模型训练操作中，需要对源代码数据进行语法解析并且解析出跨文件的依赖关系。具体的，通过对源代码进行 AST 语法解析，获得每个文件中定义的类、类中或类外包含的函数、类中包含的成员变量等会被跨文件调用的这些元信息。

在完成语法解析之后，需要解析出跨文件的依赖关系。参见图 3，图 3 是本公开一个实施例提供的一种模型训练方法的样本代码文件的示意图。其中，图 3 中的当前文件可以理解为第一样本代码文件，当前文件调用依赖的文件（file_reader）和当前文件调用依赖的文件（sorter）可以理解为第二样本代码文件。也即是说，该第一样本代码文件可以为一个或多个，每个第一样本代码文件可以存在对应的一个或多个第二样本代码文件。基于图 3 可知，为了解析出跨文件的依赖关系，需要解析当前文件的源代码中调用文件“sorter”文件或“file_reader”文件中的类和函数的调用语句，其中，“data=file_reader.read_json(data_path)”是当前文件中针对“file_reader”文件的类调用语句，“Sorter=sorter()”是针对“sorter”文件的类调用语句。基于文件之间的类和

函数的调用关系建文件之间的调用依赖关系。

此外，在解析跨文件的依赖关系的过程中，可以采用一个依赖关系分析工具实现。例如，该依赖关系分析工具可以是基于 tree-sitter 工具，该 tree-sitter 工具，能够对依赖关系进行分析。

步骤 204：基于所述第一样本代码文件，确定第一样本代码数据以及样本代码标签。

其中，第一样本代码数据可以理解为从第一样本代码文件中获取的、作为训练样本的一个或多个代码语句。在本公开提供的一个或多个实施例中，第一样本代码数据以及样本代码标签可以从该第一样本代码数据中提取，通过该步骤，实现了利用关联度较高的样本代码数据和样本代码标签进行模型训练，提高代码处理模型进行代码处理的性能。并且利用性能较高的代码处理模型对待处理代码数据进行高效的处理，从而提高了软件开发的效率，避免了由于软件开发效率较低导致的问题。此外，通过“第一样本代码数据以及样本代码标签可以从该第一样本代码数据中提取”步骤，避免了由于采用样本和标签分离的技术方案，导致训练后的模型性能较差的问题。该样本和标签分离的技术方案可以理解为将源代码文件作为样本，并为该样本添加对应的标签的技术方案，由于该方案中样本和标签互相独立且关联不高，因此会导致训练后的模型性能较差。

样本代码标签可以理解为从第一样本代码文件中获取的、作为样本标签的一个或多个代码调用语句，其中，该代码调用语句可以理解为第一样本代码文件中能够调用第二样本代码文件的调用语句。

在本公开提供的一个或多个实施例中，所述基于所述第一样本代码文件，确定第一样本代码数据以及样本代码标签，包括：

从所述第一样本代码文件中，确定调用所述第二样本代码文件的调用代码语句，并将所述调用代码语句，确定为所述样本代码标签；

从所述第一样本代码文件中，确定除所述样本代码标签之外的其他代码语句，并将所述其他代码语句确定为所述第一样本代码数据。

其中，调用代码语句可以理解为第一样本代码文件中涉及跨文件调用第二样本代码文件的代码语句。例如“data =file_reader.read_json(data_path)”，其中，file_reader 可以是被调用的源代码文件（即第二样本代码文件）。

沿用上例，其中，该代码处理模型可以是一种修复代码自动补全模型。基于此，为了构造训练数据，基于图 3 可知，本模型训练方法首先采样一条当前文件（第一样本代码文件）中涉及跨文件调用的语句“data =file_reader.read_json(data_path)”，这是一条调用语句。其次，在确定该调用语句之后，可以将该调用语句作为样本代码标签。

具体的，在解析当前文件中的调用语句的过程中，调用语句分成 2 类：类调用语句和函数调用语句。其中，类调用语句主要是类的定义部分。其中，函数调用语句主要是调用其他文件中的一些函数获得想要的输出结果，目的是从其他文件中获取内容，例如，一些调用语句的目的是获得冒泡排序和快速排序的排序结果。这些调用语句涉及类和函数的跨文件调用，可以作为模型训练过程的目标（样本代码标签）。

然后，将当前文件中该调用语句上方的代码内容作为代码上文，将该调用语句下方的代码内容作为代码下文，并且，代码上文和代码下文作为训练样本（第一样本代码数据）。获取该第一样本代码数据和样本代码标签进行模型训练之后，模型的预测目标是预测完整的调用语句。

因此，通过获得第一样本代码数据和样本代码标签，实现了利用关联度较高的样本代码数据和样本代码标签进行模型训练，提高代码处理模型进行代码处理的性能。并且利用性能较高的代码处理模型对待处理代码数据进行高效的处理，从而提高了软件开发的效率，避免了由于软件开发效率较低导致的问题。

步骤 206：基于所述第二样本代码文件确定第二样本代码数据，并基于所述第一样本

代码数据以及所述第二样本代码数据，确定样本代码数据。

其中，第二样本代码数据可以理解为作为训练样本的调用代码数据，该调用代码数据可以理解为源代码文件中被其他源代码文件所调用的数据/变量。例如，该调用代码数据可以为源代码文件中定义的类、类中包含的函数、类外包含的函数、或者类中包含的成员变量等会被跨文件调用的数据。

样本代码数据可以理解为作为训练样本的代码数据，用于对代码处理模型进行训练。该样本代码数据包括第一样本代码数据以及第二样本代码数据。

在本公开提供的一个或多个实施例中，所述基于所述第二样本代码文件确定第二样本代码数据，包括：

从所述第二样本代码文件中，确定初始调用代码数据；

对所述初始调用代码数据进行数据筛选处理，获得目标调用代码数据，并将所述目标调用代码数据，作为所述第二样本代码数据。

具体的，本公开一个或多个实施例中提供的模型训练方法，在进行模型训练数据构建的过程中，针对该第二样本代码数据具有两个阶段的处理步骤，首先，从第二样本代码文件获取初始调用代码数据。然后，对初始调用代码数据进行数据筛选处理，获得数据筛选后的目标调用代码数据，并将目标调用代码数据作为第二样本代码数据，从而利用数据筛选后的调用代码数据进行模型训练，从而进一步提高代码处理的性能。基于此，该第二样本代码数据是从第二样本代码文件中获取的。

在本公开提供的一个或多个实施例中，所述从所述第二样本代码文件中，确定初始调用代码数据，包括：

确定所述第二样本代码文件中的代码数据，并对所述代码数据进行语法分析，获得所述代码数据中的多个调用代码数据；

将各调用代码数据进行数据拼接，获得所述初始调用代码数据。

其中，代码数据可以理解为第二样本代码文件中的源代码。该语法分析可以理解为对源代码数据进行语法解析，该语法解析可以基于 AST 实现。

沿用上例，当前文件中可以分别导入了“fromfile_readerimportFileReader”和“fromsorterimportSorter”，也就是说，存在 file_reader.py 到 main.py 的调用依赖和 sorter.py 到 main.py 的调用依赖。其中，file_reader.py 和 sorter.py 可以理解为第二样本代码文件。因此，需要分别解析 file_reader.py 和 sorter.py 中定义的类/函数/成员变量等信息。

基于此，利用 AST 解析 file_reader.py 和 sorter.py 两个文件，获得调用依赖解析出来的元数据，如图 3 中的调用数据，解析出来的元数据包含 file_reader.py 中的 FileReader 类，该类包含多个函数并且解析出这多个函数的定义语句，元数据还包含 sorter.py 中的 Sorter 类，该类包含多个函数并且解析出这多个函数的定义语句。

此外，图 3 中的调用数据可以理解为从第二样本代码文件中解析出来的第二样本代码数据。

最后，将所有解析出来的数据拼接在一起，最终得到当前文件所依赖的所有内容的元数据，也就是说，调用依赖元数据（即目标调用代码数据）包含了当前文件所依赖的其他文件中的类/函数/成员变量的所有信息，而这些信息是解决调用幻觉的基础。

在本公开提供的一个或多个实施例中，在构造调用依赖元数据<import>（即目标调用代码数据）时，会存在一些需要注意的问题如下：1、当前文件中，导入语句部分可能非常多，但是可能在源代码中，并不是所有导入的信息都被真正调用了。2、依赖的文件（第二样本代码文件）中，可能定义的类和函数数量非常多，但这些类和函数大多数也不会真正被调用到。

因为存在这两方面问题，最终可能导致构造出来的调用依赖元数据内容非常多，这不仅会对模型预测带来很大的干扰，而且会导致上下文长度急剧扩张，使得推理速度变慢。为了解决这方面的问题，本公开一个或多个实施例提供的模型训练方法提出采用负采样的方式，在保证真正调用语句对应的定义信息出现在依赖元数据中，而且不会造成依赖元数据的上下文长度过长。具体过程如下。

对所述初始调用代码数据进行数据筛选处理，获得目标调用代码数据，包括：

从所述初始调用代码数据中，确定第一调用代码数据、第二调用代码数据以及第三调用代码数据，其中，所述第一调用代码数据为被所述第一样本代码文件调用的数据，所述第二调用代码数据为被导入所述第二样本代码文件中的数据，所述第三调用代码数据为被全局定义的数据；

确定所述第一调用代码数据的第一数据数量；

基于所述第一数据数量以及预设数量比例，从所述第二调用代码数据以及所述第三调用代码数据中，选择与所述第一调用代码数据对应的第四调用代码数据；

将所述第一调用代码数据以及所述第四调用代码数据，确定为所述目标调用代码数据。

其中，第一调用代码数据可以理解为源代码文件的所有调用语句中一定被调用的类/函数/成员变量。第二调用代码数据可以理解为源代码文件的所有导入语句中，没有被调用、但是被导入的类/函数/成员变量。第三调用代码数据可以理解为源代码文件的所有导入语句中，没有被调用、没有被导入，但是被定义的类/函数/成员变量。

其中，第一数据数量可以理解为第一调用代码数据对应的数据个数。在第一调用代码数据为调用依赖元数据的情况下，第一数据数量可以是调用依赖元数据中元数据的个数。预设数量比例可以根据实际应用场景进行设置，本公开对此不作具体限制。

沿用上例，对调用依赖元数据进行负采样的方式为：解析当前文件（即第二样本调用文件）对应的项目下的所有定义好的类/函数/成员变量；解析当前文件中所有导入语句，获得可能导入的类/函数/成员变量；解析当前文件中所有调用语句，获得一定被调用的类/函数/成员变量，作为正向数据，存入调用依赖元数据中；采样一些没有被调用但是被导入的类/函数/成员变量，作为负向数据，存入调用依赖元数据中；采样一些没有被导入但是被定义的类/函数/成员变量，作为负向数据，存入调用依赖元数据中；需要说明的是，在通过负采样的方式对调用依赖元数据进行优化的过程中，控制正向数据和负向数据的比例为 1:5（即预设数量比例）左右；

通过上述步骤最终获得的调用依赖元数据，不仅包含了一定被调用的类/函数/成员变量，还包含被导入但是没有调用和被定义的一些类/函数/成员变量，这样在正负数据同时存在的情况下，可以帮助代码处理模型学习参考调用依赖元数据的信息，从而辅助编写后续的调用语句，尽可能地消除调用幻觉问题。

步骤 208：基于所述样本代码数据以及所述样本代码标签，对所述待训练代码处理模型进行模型训练，获得训练完成的代码处理模型。

在本公开提供的一个或多个实施例中，所述基于所述样本代码数据以及所述样本代码标签，对所述待训练代码处理模型进行模型训练，获得训练完成的代码处理模型，包括：

将所述样本代码数据输入所述待训练代码处理模型，获得样本代码数据对应的目标代码数据；

基于所述样本代码数据对应的目标代码数据以及所述样本代码标签，确定损失值；

基于所述损失值对所述待训练代码处理模型进行模型训练，直至达到模型训练停止条件，获得训练完成的所述代码处理模型。

沿用上例，利用训练样本对代码自动补全模型进行训练，其训练方式为：首先，将训练样本（样本代码数据）输入至代码自动补全模型中，获得模型输出的预测调用语句（样本

代码数据对应的目标代码数据）；其次，利用预测调用语句和样本标签（样本代码标签）计算损失值；具体的，本公开一个或多个实施例中的模型训练方法，可以根据实际应用场景采用损失函数计算损失值，该损失函数可以为均方误差损失函数，该损失函数可以计算模型预测值（即预测调用语句）与真实目标值（即样本标签）之间的平均平方距离，并将该平均平方距离作为损失值。需要说明的是，在本公开提供的一个或多个实施例中，损失值的计算可以按照前向传播、计算损失、求平均、反向传播与优化等步骤计算获得。

最后，基于损失值对代码自动补全模型进行模型参数调整，直至达到模型训练停止条件，获得训练完成的代码自动补全模型。其中，模型训练停止条件可以根据实际场景进行设置，例如，模型训练达到预设轮次，损失值达到收敛等。

基于此，在获得训练完成的代码处理模型的情况下，后续可以利用训练好的该代码处理模型，对软件开发过程中的待处理代码数据进行处理，从而提高了软件开发的效率，避免了软件开发过程中时间成本和人力成本被大量浪费的问题。

在本公开提供的一个或多个实施例中，所述基于所述样本代码数据以及所述样本代码标签，对所述待训练代码处理模型进行模型训练，获得训练完成的所述代码处理模型，包括：

利用训练数据模板对所述样本代码数据以及所述样本代码标签进行处理，获得模型训练数据；

基于所述模型训练数据对所述待训练代码处理模型进行模型训练，获得训练完成的所述代码处理模型。

其中，训练数据模板可以理解为将不同的训练数据转换为模型训练时常见的形式的模板。该训练数据模板可以为 prompt。

沿用上例，在获得样本代码数据以及样本代码标签之后，需要基于样本代码数据以及样本代码标签构造训练数据。具体的，针对采样大量的包含跨文件调用语句获得代码块，将该代码块的上文代码（即代码前文）和下文代码（即代码后文）作为上下文，其目标是模型正确地写出调用语句，同时，需要将包含该调用语句定义部分的内容，加入到 prompt 中，从而消除模型训练过程中的调用幻觉问题。也即是说，将上述实施例中解析出来的调用依赖元数据也作为额外信息放入 prompt 中。例如，最终组织成如下的 prompt（训练数据）可以为：<filename>当前文件名<import>调用依赖元数据<fim_prefix>代码前文<fim_suffix>代码后文<fim_middle>目标调用语句，具体可以参见图 4，图 4 是本公开一个实施例提供的一种模型训练方法的 prompt 的示意图。

在本公开提供的一个或多个实施例中，所述模型训练方法还包括：

确定待处理代码数据；

将所述待处理代码数据输入所述训练完成的代码处理模型，利用所述训练完成的代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作。

其中，该待处理代码数据可以理解为需要通过代码处理模型进行处理的代码数据，该待处理代码数据可以为一个或多个代码语句，例如“data_p**th ='data. j**on' ”。或者，该待处理代码数据可以为一个或多个代码文件。在本公开提供的一个或多个实施例中，该待处理代码数据可以待补全代码数据，该待补全代码数据可以理解为并未编写完成的一个或多个待补全代码语句，或者并未编写完成的待补全代码文件；例如，“data_p**th = ”，该待补全代码语句中存在没有编写完成的部分内容。

该代码补全操作可以理解为将未编写完成的待补全代码数据进行代码补全的操作。

代码生成操作可以理解为对于已经编写的部分代码，为其生成相对应的后续代码数据的操作；该代码生成操作可以根据一个未编写完成的代码文件中的部分代码，生成对应的后续代码，从而获得编写完成的完整代码文件。

代码修复操作可以理解为对软件开发过程中需要修复的代码数据进行修复的操作。例如，对于一个编写完成的、但代码语句存在缺陷的代码数据，可以通过代码修复操作对其进行修复，从而获得一个编写完成的、且代码语句不存在缺陷的代码数据。

在本公开提供的一个或多个实施例中，所述确定待处理代码数据，包括：

获取用户通过客户端编写的待处理代码数据；

其中，该用户可以通过代码编写系统对待处理代码数据进行编码，在编写过程中，可以获得该用户编写的待处理代码数据。

其中，该代码编写系统可以理解为智能编码助手，智能编码助手可以基于先进的人工智能技术，通过深入理解开发者的编程意图和上下文信息，能够实现代码自动补全、代码片段推荐、错误检测与修复等功能，显著提升开发者的编码效率。本公开一个或多个提供的代码处理模型，可以部署在代码编写系统中，为开发者的开发工作提供帮助，提高开发效率。

具体的，获取用户编写的待处理代码数据可以理解为对用户编写的待处理代码数据进行检测，从而获得该待处理代码数据。或者，在用户编写待处理代码数据之后，可以通过客户端中的任务处理界面将该待处理代码数据提供至服务端，从而让服务端获得待处理代码数据。从而便于后续利用代码处理模型对该待处理代码数据进行处理，提高软件开发效率。该服务端可以理解为本公开一个或多个实施例中的代码处理方法所应用的服务端。

本公开提供的一个或多个实施例中，在通过样本代码数据和样本代码标签获得训练完成的代码处理模型之后，将待处理代码数据输入该代码处理模型，获得待处理代码数据对应的目标代码数据。

在本公开提供的一个或多个实施例中，所述将所述待处理代码数据输入代码处理模型，获得所述待处理代码数据对应的目标代码数据之后，还包括：

将所述目标代码数据发送至所述客户端，使所述客户端将目标代码数据提供至所述用户。

具体的，在将用户编写的待处理代码数据输入该代码处理模型，获得待处理代码数据对应的目标代码数据之后，可以将该目标代码数据发送至所述客户端，使客户端将目标代码数据提供给用户，从而便于用户基于该目标代码数据进行高效率的软件开发工作，提高软件开发效率。

本公开一个或多个实施例中的模型训练方法，可以确定第一样本代码文件以及与第一样本代码文件存在调用关系的第二样本代码文件，并且，通过基于第一样本代码文件确定的第一样本代码数据以及样本代码标签、基于第二样本代码文件确定第二样本代码数据进行模型训练，实现在模型训练过程中，利用彼此之间存在调用关系的样本代码文件对代码处理模型进行训练，从而使得训练后的代码处理模型，能够对实际开发场景中调用关系较为复杂的代码数据进行处理，提高了实际场景中代码处理模型的性能；并且，在将性能较好的代码处理模型应用于实际开发场景中，能够提高软件开发的效率，实现高效的完成软件开发，从而避免了无法及时的利用开发的软件提供服务的问题。

下述结合附图 5，以本公开提供的模型训练方法在修复代码自动补全模型的跨文件调用幻觉场景的应用为例，对所述模型训练方法进行进一步说明。其中，图 5 示出了本公开一个实施例提供的一种模型训练方法的处理过程流程图，具体包括以下步骤。

步骤 502：收集源代码数据。

具体的，此步骤中会收集大量的源代码数据，用于进行模型训练。其中，该源代码数据通过两种方式收集；

一种方式是：针对各种领域、各种程序产品的开发人员，采集他们在日常工作中所编写的源代码数据，从而使得训练过程中的训练数据与用户真实开发场景接近。

另一种方式是：从 The-Stack 数据集中确定源代码数据，该数据集主要爬取并处理了 Github 中的海量仓库，包含各种语言的项目的源代码数据，共包含 3T 左右的数据。

步骤 504：解析跨文件的依赖关系。

首先，对收到的源代码数据进行 AST 语法解析，获得当前文件（是指编写的一个源代码数据文件）中定义的类、类中或类外包含的函数、类中包含的成员变量等会被跨文件调用的编码数据；

其次，将文件所有解析出来的数据拼接在一起，最终得到该文件所依赖的所有内容的元数据。

需要说明的是，通过对每个文件执行上述操作，从而确定每个文件的元数据（即调用数据，或调用依赖元数据）。

最后，解析出跨文件的依赖关系。

具体的，解析依赖关系可以通过两种方式实现。

一种方式是：

首先，根据步骤 504 解析的元数据，确定 A 文件中所调用到的 B 文件中的类和函数；

其次，基于 A 文件和 B 文件之间调用的类和函数，确定建立 B 文件到 A 文件之间的调用依赖关系。

另一种方式是：基于 tree-sitter 工具，确定 A 文件和 B 文件之间的调用依赖关系。

其中，tree-sitter 是一个解析器生成工具和增量解析库，能够将代码源文件解析成具体的语法树，从而对绝大多数语言进行解析。

步骤 506：采样调用语句构造训练数据。

该步骤的具体执行方式为：

1、确定采样一条当前文件中涉及跨文件调用的语句，将其作为目标调用语句；

2、将当前文件中，该目标调用语句上方的内容作为代码上文，将该目标调用语句下方的内容作为代码下文；

3、采用负采样的方法对元数据进行优化，获得优化后的元数据。

由于一些问题，可能导致构造出来的调用依赖元数据内容非常多，从而对模型预测带来很大的干扰，并且导致上下文长度急剧扩张，使得推理速度变慢。

基于此，采用负采样的方法对元数据进行优化，具体实现方式包括：

①解析当前文件对应的项目下的所有定义好的类/函数/成员变量。

②解析当前文件中所有导入语句，获得可能导入的类/函数/成员变量。

③解析当前文件中所有调用语句，获得一定被调用的类/函数/成员变量，作为正向数据，存入调用依赖元数据中。

④采样一些没有被调用但是被导入的类/函数/成员变量，作为负向数据，存入调用依赖元数据中。

⑤采样一些没有被导入但是被定义的类/函数/成员变量，作为负向数据，存入调用依赖元数据中。

需要说明的是，在优化过程中，控制正向数据和负向数据的比例为 1:5 左右。

4、将目标调用语句作为样本标签，将代码上文、代码下文、优化后的元数据作为训练样本，从而完成训练数据的构建。

5、将训练样本和标签放入 Prompt 中，构造一条训练数据。

步骤 508：训练模型。

该步骤的具体执行方式为：

首先，将训练样本输入至代码自动补全模型中，获得模型输出的预测调用语句；

其次，利用预测调用语句和样本标签（目标调用语句）计算损失值；

最后，基于损失值对代码自动补全模型进行模型参数调整，直至达到模型训练停止条件，获得训练完成的代码自动补全模型。

此外，需要说明的是，代码自动补全模型在训练完成之后，可以将其应用到代码补全场景中，具体应用的步骤包括：

首先，获取程序人员编写的待补全代码数据；

其次，将待补全代码数据输入训练后的代码自动补全模型，获得代码自动补全模型输出的预测代码（代码补全数据），其中，该预测代码与待补全代码数据相对应，能够对待补全代码数据进行补全。

最后，将预测代码向用户展示，让用户使用代码自动补全模型生成的预测代码。

基于上述步骤可知，本公开一个或多个实施例提供的模型训练方法，提出了一种修复代码自动补全模型的跨文件调用幻觉的方法，通过对原始的代码数据进行 AST 解析跨文件类或函数的调用依赖，获取调用依赖元数据，并将当前文件中包含的跨文件调用定义语句和调用依赖元数据放入 prompt 进行训练，从而消除模型的跨文件调用的幻觉问题。

此外，本公开一个或多个实施例提供的模型训练方法，提出了基于 AST 语法解析，得到跨文件调用的调用依赖元数据，该元数据（即调用依赖元数据）对跨文件的类、函数和成员变量进行了信息整合，从而缓解调用幻觉问题。

本公开一个或多个实施例提供的模型训练方法，提出了将跨文件调用的信息总结成包含定义语句的调用依赖元数据，并且通过负采样技术进行训练，大幅缩短了上下文长度，并且使得训练和预测的没有 gap，因此可以解决缓解调用幻觉问题。

以及，本公开一个或多个实施例提供的模型训练方法，提出一种对调用依赖元数据的获取方案，并且通过负采样技术对调用依赖元数据的信息进行整合，加入一些没有被调用的类和函数，从而帮助提升模型学习正确地调用元数据中定义的类和函数。

参见图 6，图 6 示出了根据本公开一个实施例提供的一种代码处理方法的流程图，具体包括以下步骤。

步骤 602：接收用户通过所述客户端中的代码处理界面发送的待处理代码数据。

步骤 604：将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根据上述模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作。

步骤 606：将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示。

其中，该代码处理界面可以理解为展示在客户端上、用于让用户进行代码处理的界面。该代码处理界面可以为应用程序界面、互联网页面等。该代码处理界面中包括多种类型的代码处理控件（例如按钮、文本框），用户可以通过该代码处理控件对待处理代码数据进行处理。例如，在文本框中输入待处理代码数据，并通过点击代码数据处理按钮，将待处理代码数据提供至所述代码处理方法所应用的服务端，从而通过所述代码处理方法对该待处理代码进行处理。

在本公开提供的一个或多个实施例中，所述待处理代码数据为待补全代码数据；

所述将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，包括：

将所述待补全代码数据输入所述代码处理模型，利用所述代码处理模型对所述待补全代码数据进行所述代码补全操作，获得所述待补全代码数据对应的代码补全数据；

所述将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示，包括：

将所述代码补全数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述代码补全数据向所述用户展示。

其中，待补全代码数据可以理解为并未编写完成的一个或多个待补全代码语句，或者并未编写完成的待补全代码文件，例如，代码语句“data_p**th = ”是一个未编写完成的代码语句。通过代码处理模型对其进行 d 代码补全操作之后，可以获得对应的代码补全数据，例如，代码语句“data_p**th ='data. j**on'”，可以是上述待补全代码数据对应的代码补全数据。

在本公开提供的一个或多个实施例中，所述待处理代码数据为待修复代码数据；

所述将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，包括：

将所述待修复代码数据输入所述代码处理模型，利用所述代码处理模型对所述待修复代码数据进行所述代码修复操作，获得所述待修复代码数据对应的代码修复数据；

所述将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示，包括：

将所述代码修复数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述代码修复数据向所述用户展示。

其中，待修复代码数据可以理解为在代码编写过程中存在缺陷的一个或多个待修复代码语句，或者在代码编写过程中存在缺陷的待修复代码文件。该代码修复数据可以理解为通过代码处理模型对该待修复代码数据进行代码修复操作后获得的一个或多个修复完成的代码语句或者修复完成的代码文件。在代码编写过程中可能会存在部分语句发生拼写错误、代码逻辑错误等缺陷。对于这些缺陷，可以将存在该缺陷的待修复代码数据输入代码处理模型，获得代码修复数据。例如，代码语句“data_='data. j**on'”存在拼写错误问题，该代码语句中的“data_p**th”缺少了“p**th”。通过将该代码语句输入至代码处理模型中进行代码修复操作，从而获得完整且正确的代码语句“data_p**th ='data. j**on'”。

需要说明的是，该客户端可以理解为终端。

本公开一个或者多个实施例中提供的代码处理方法，在接收到用户通过客户端中的代码处理界面发送的待处理代码数据之后，能够利用该代码处理模型对该待处理代码数据进行代码补全操作、代码生成操作或者代码修复操作等数据调整操作，获得目标代码数据，从而实现了在软件开发过程中为用户的代码编写工作提供帮助，从而提高了软件开发的效率，节约了软件开发过程中时间成本和人力成本，并且，由于采用代码处理模型对待处理代码数据进行处理，避免了开发者使用计算机设备进行软件开发的过程中，由于软件开发效率较低导致大量计算机设备的计算资源被浪费的问题。

需要说明的是，该代码处理方法的技术方案与上述的模型训练方法的技术方案属于同一构思，代码处理方法的技术方案未详细描述的细节内容，均可以参见上述模型训练方法的技术方案的描述。

参见图 7，图 7 示出了根据本公开一个实施例提供的一种代码处理平台的结构示意图，该代码处理平台 700 包括请求接收单元 702、代码处理单元 704 和响应单元 706；

所述请求接收单元 702，用于接收代码处理请求，并基于所述代码处理请求确定待处理代码数据以及所述待处理代码数据对应的代码处理模型标识；

所述代码处理单元 704，用于基于所述代码处理模型标识确定代码处理模型，将所述待处理代码数据输入所述代码处理模型，利用所述代码处理模型对所述待处理代码数据进

行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根据上述模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作；

所述响应单元 706，用于输出所述目标代码数据。

其中，该代码处理平台可以理解为对待处理代码数据进行处理的平台，该代码处理平台在软件开发过程中为代码编写提供相关的服务。开发人员可以在所述代码处理平台中进行代码编写。在本公开提供的一个或多个实施例中，代码处理平台可以理解为智能编码助手，智能编码助手可以基于先进的人工智能技术，通过深入理解开发者的编程意图和上下文信息，能够实现代码自动补全、代码片段推荐、错误检测与修复等功能，显著提升开发者的编码效率。

该请求接收单元 702 可以理解为代码处理平台中用于接收代码处理请求的模块，本公开一个或多个实施例中提供的代码处理平台中，用户通过客户端的代码处理界面向代码处理平台发送代码处理请求的情况下，是由代码处理平台中的请求接收单元 702 对该代码处理请求进行接收；并且，还可以从该代码处理请求中确定待处理代码数据以及该代码处理模型标识。

其中，该代码处理模型标识可以为唯一标识代码处理模型的信息，例如，该代码处理模型标识可以为代码处理模型名称、编号等信息，或者该代码处理模型标识可以为代码处理模型对应的代码处理模型地址。

其中，代码处理单元 704 可以理解为代码处理平台中用于对待处理代码数据进行代码补全操作、代码生成操作或者代码修复操作等代码调整操作的单元；该代码调整操作是利用代码处理模型实现。

其中，该响应单元 706 可以理解为代码处理平台中用于对外输出目标代码数据的模块。当代码处理单元 704 利用代码处理模型获得待处理代码数据对应的目标代码数据之后，由该响应单元 706 对外输出目标代码数据。

需要说明的是，本公开一个或多个实施例中提供的代码处理平台中，该请求接收单元 702、代码处理单元 704 和响应单元 706 可以为硬件模块或者软件模块。

在本公开提供的一个或多个实施例中，所述请求接收单元 702，用于从所述代码处理请求中，确定所述待处理代码数据以及所述待处理代码数据对应的所述代码处理模型标识；

具体的，代码处理平台在进行代码处理的过程中，接收到的代码处理请求中会携带有待处理代码数据以及待处理代码数据对应的代码处理模型标识。在此情况下，可以从代码处理请求中确定待处理代码数据以及待处理代码数据对应的代码处理模型标识，然后将该待处理代码数据以及代码处理模型标识发送至代码处理单元 704，使代码处理单元 704 基于该代码处理模型标识对应的代码处理模型，对该待处理代码数据进行代码调整操作，获得目标代码数据。

在本公开提供的一个或多个实施例中，代码处理平台还包括代码问答功能，用户可以通过客户端上的代码处理界面向代码处理平台输入代码生成文本。代码处理平台基于该代码生成文本，生成对应的代码数据，从而满足用户的实际应用需求。具体的，所述请求接收单元 702，用于确定所述代码处理请求中的代码生成文本以及所述代码生成文本对应的处理提示信息，基于所述处理提示信息将所述代码生成文本输入代码生成模型，获得所述待处理代码数据，并确定所述待处理代码数据对应的所述代码处理模型标识。

其中，代码生成文本可以理解为用于生成代码的文本，该代码生成文本可以理解为是对需要生成的代码数据的代码描述文本。例如，用户可以输入“生成抽奖代码”这一文本数据，而代码处理平台可以根据“生成抽奖代码”，生成对应的抽奖代码数据。

处理提示信息可以理解为指示代码处理平台对代码生成文本进行何种操作的提示信息。由于代码处理平台提供了多种针对代码数据的处理功能，用来满足软件开发工作的需

要。基于此，代码处理平台需要基于该处理提示信息对代码生成文件进行处理，避免由于针对代码生成文件的错误处理，导致无法满足用户需求的问题发生。该处理提示信息可以是对代码生成文本的处理过程的描述信息。例如，该处理提示信息可以为：首先利用代码生成模型生成代码生成文本对应的待处理代码数据，然后利用代码处理模型标识对应的代码处理模型，对该待处理代码数据进行代码调整操作，获得目标代码数据。

代码生成模型可以理解为基于代码生成文本生成对应的代码数据的模型。例如，将“生成抽奖代码”这一文本数据输入代码生成模型，利用该代码生成模型生成与“生成抽奖代码”对应的抽奖代码数据。

以代码处理平台在代码问答场景下为例，对代码处理平台进行解释说明。其中，代码生成文本可以为“生成抽奖代码”。基于此，用户通过客户端上的代码处理界面向代码处理平台发送代码处理请求，该代码处理请求中携带有代码生成文本以及所述代码生成文本对应的处理提示信息。该请求接收单元 702，在接收到所述代码处理请求之后，获取代码处理请求中的代码生成文本“生成抽奖代码”以及对应的处理提示信息。然后基于处理提示信息将“生成抽奖代码”输入代码生成模型，获得抽奖代码数据。

在获得抽奖代码数据之后，为了保证代码数据的正确性、或者保证代码数据的完整性。需要从处理提示信息中确定对该抽奖代码数据进行代码补全操作或者代码修复操作的代码处理模型的代码处理模型标识。并将该代码处理模型标识和抽奖代码数据发送至代码处理单元 704。

代码处理单元 704 会确定代码处理模型标识对应的代码处理模型，并利用代码处理模型对抽奖代码数据进行代码补全操作或者代码修复操作，从而获得具备正确性和完整性的抽奖代码数据。

在获得具备正确性和完整性的抽奖代码数据之后，通过响应模块将该抽奖代码数据发送至客户端，使客户端通过代码处理界面将该抽奖代码数据展示给用户。从而实现了在软件开发过程中根据代码生成文本生成对应的目标代码数据，避免了用户手动编写代码导致的代码编写效率低、以及手动编写代码的错误率较高的问题，提高了软件开发的效率。

本公开一个或多个实施例中提供的代码处理平台，请求接收单元能够基于接收到的代码处理请求，确定待处理代码数据以及待处理代码数据对应的代码处理模型标识，该代码处理单元，能够利用代码处理模型对待处理代码数据进行代码补全操作、代码生成操作或者代码修复操作等数据调整操作，获得目标代码数据。最后，通过响应单元输出目标代码数据；从而通过各单元之间的相互配合，实现了在软件开发过程中为用户的代码编写工作提供帮助，从而提高了软件开发的效率，节约了软件开发过程中时间成本和人力成本，并且，由于采用代码处理模型对待处理代码数据进行处理，避免了开发者使用计算机设备进行软件开发的过程中，由于软件开发效率较低导致大量计算机设备的计算资源被浪费的问题。

需要说明的是，该代码处理平台的技术方案与上述的代码处理方法或模型训练方法的技术方案属于同一构思，代码处理平台的技术方案未详细描述的细节内容，均可以参见上述代码处理方法或模型训练方法的技术方案的描述。

与上述方法实施例相对应，本公开还提供了模型训练装置实施例，图 8 示出了本公开一个实施例提供的一种模型训练装置的结构示意图。如图 8 所示，该装置包括：

代码文件确定模块 802，被配置为确定待训练代码处理模型对应的第一样本代码文件以及第二样本代码文件，其中，所述第一样本代码文件与所述第二样本代码文件存在调用关系；

第一样本确定模块 804，被配置为基于所述第一样本代码文件，确定第一样本代码数据以及样本代码标签；

第二样本确定模块 806，被配置为基于所述第二样本代码文件确定第二样本代码数据，

并基于所述第一样本代码数据以及所述第二样本代码数据，确定样本代码数据；

模型训练模块 808，被配置为基于所述样本代码数据以及所述样本代码标签，对所述待训练代码处理模型进行模型训练，获得训练完成的代码处理模型。

可选地，所述模型训练装置还包括代码处理模块，被配置为：

确定待处理代码数据；

将所述待处理代码数据输入所述训练完成的代码处理模型，利用所述训练完成的代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作。

可选地，所述第一样本确定模块 804，还被配置为：

从所述第一样本代码文件中，确定调用所述第二样本代码文件的调用代码语句，并将所述调用代码语句，确定为所述样本代码标签；

从所述第一样本代码文件中，确定除所述样本代码标签之外的其他代码语句，并将所述其他代码语句确定为所述第一样本代码数据。

可选地，所述第二样本确定模块 806，还被配置为：

从所述第二样本代码文件中，确定初始调用代码数据；

对所述初始调用代码数据进行数据筛选处理，获得目标调用代码数据，并将所述目标调用代码数据，作为所述第二样本代码数据。

可选地，所述第二样本确定模块 806，还被配置为：

确定所述第二样本代码文件中的代码数据，并对所述代码数据进行语法分析，获得所述代码数据中的多个调用代码数据；

将各调用代码数据进行数据拼接，获得所述初始调用代码数据。

可选地，所述第二样本确定模块 806，还被配置为：

从所述初始调用代码数据中，确定第一调用代码数据、第二调用代码数据以及第三调用代码数据，其中，所述第一调用代码数据为被所述第一样本代码文件调用的数据，所述第二调用代码数据为被导入所述第二样本代码文件中的数据，所述第三调用代码数据为被全局定义的数据；

确定所述第一调用代码数据的第一数据数量；

基于所述第一数据数量以及预设数量比例，从所述第二调用代码数据以及所述第三调用代码数据中，选择与所述第一调用代码数据对应的第四调用代码数据；

将所述第一调用代码数据以及所述第四调用代码数据，确定为所述目标调用代码数据。

可选地，所述代码文件确定模块 802，还被配置为：

确定待训练代码处理模型对应的多个样本代码文件，并对各样本代码文件进行语法分析，获得各样本代码文件中的调用代码数据；

基于所述调用代码数据，从所述各样本代码文件中确定所述第一样本代码文件以及与所述第一样本代码文件存在调用关系的所述第二样本代码文件。

可选地，所述模型训练模块 808，还被配置为：

利用训练数据模板对所述样本代码数据以及所述样本代码标签进行处理，获得模型训练数据；

基于所述模型训练数据对所述待训练代码处理模型进行模型训练，获得所述训练完成的代码处理模型。

本公开一个或多个实施例中提供的模型训练装置，可以确定第一样本代码文件以及与第一样本代码文件存在调用关系的第二样本代码文件，并且，通过基于第一样本代码文件

确定的第一样本代码数据以及样本代码标签、基于第二样本代码文件确定第二样本代码数据进行模型训练，实现在模型训练过程中，利用彼此之间存在调用关系的样本代码文件对代码处理模型进行训练，从而使得训练后的代码处理模型，能够对实际开发场景中调用关系较为复杂的代码数据进行处理，提高了实际场景中代码处理模型的性能；并且，在将性能较好的代码处理模型应用于实际开发场景中，能够提高软件开发的效率，实现高效的完成软件开发，从而避免了无法及时的利用开发的软件提供服务的问题。

上述为本实施例的一种模型训练装置的示意性方案。需要说明的是，该模型训练装置的技术方案与上述的模型训练方法的技术方案属于同一构思，模型训练装置的技术方案未详细描述的细节内容，均可以参见上述模型训练方法的技术方案的描述。

与上述方法实施例相对应，本公开还提供了代码处理装置实施例，图 9 示出了本公开一个实施例提供的一种代码处理装置的结构示意图。如图 9 所示，该装置包括：

代码数据接收模块 902，被配置为接收用户通过所述客户端中的代码处理界面发送的待处理代码数据；

代码数据处理模块 904，被配置为将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根据上述模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作；

代码数据发送模块 906，被配置为将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示。

可选地，所述待处理代码数据为待补全代码数据；

所述代码数据处理模块 904，还被配置为：

将所述待补全代码数据输入所述代码处理模型，利用所述代码处理模型对所述待补全代码数据进行所述代码补全操作，获得所述待补全代码数据对应的代码补全数据；

所述代码数据发送模块 906，还被配置为：

将所述代码补全数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述代码补全数据向所述用户展示。

可选地，所述待处理代码数据为待修复代码数据；

所述代码数据处理模块 904，还被配置为：

将所述待修复代码数据输入所述代码处理模型，利用所述代码处理模型对所述待修复代码数据进行所述代码修复操作，获得所述待修复代码数据对应的代码修复数据；

所述代码数据发送模块 906，还被配置为：

将所述代码修复数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述代码修复数据向所述用户展示。

本公开一个或多个实施例中提供的代码处理装置，在接收到用户通过客户端中的代码处理界面发送的待处理代码数据之后，能够利用该代码处理模型对该待处理代码数据进行代码补全操作、代码生成操作或者代码修复操作等数据调整操作，获得目标代码数据，从而实现了在软件开发过程中为用户的代码编写工作提供帮助，从而提高了软件开发的效率，节约了软件开发过程中时间成本和人力成本，并且，由于采用代码处理模型对待处理代码数据进行处理，避免了开发者使用计算机设备进行软件开发的过程中，由于软件开发效率较低导致大量计算机设备的计算资源被浪费的问题。

上述为本实施例的一种代码处理装置的示意性方案。需要说明的是，该代码处理装置的技术方案与上述的代码处理方法的技术方案属于同一构思，代码处理装置的技术方案未详细描述的细节内容，均可以参见上述代码处理方法的技术方案的描述。

图 10 示出了根据本公开一个实施例提供的一种计算设备 1000 的结构框图。该计算设

备1000的部件包括但不限于存储器1010和处理器1020。处理器1020与存储器1010通过总线1030相连接，数据库1050用于保存数据。

计算设备1000还包括接入设备1040，接入设备1040使得计算设备1000能够经由一个或多个网络1060通信。这些网络的示例包括公用交换电话网（PSTN，Public Switched Telephone Network）、局域网（LAN，Local Area Network）、广域网（WAN，Wide Area Network）、个域网（PAN，Personal Area Network）或诸如因特网的通信网络的组合。接入设备1040可以包括有线或无线的任何类型的网络接口（例如，网络接口卡（NIC，network interface controller））中的一个或多个，诸如IEEE802.11无线局域网（WLAN，Wireless Local Area Network）无线接口、全球微波互联接入（Wi-MAX，Worldwide Interoperability for Microwave Access）接口、以太网接口、通用串行总线（USB，Universal Serial Bus）接口、蜂窝网络接口、蓝牙接口、近场通信（NFC，Near Field Communication）。

在本公开的一个实施例中，计算设备1000的上述部件以及图10中未示出的其他部件也可以彼此相连接，例如通过总线。应当理解，图10所示的计算设备结构框图仅仅是出于示例的目的，而不是对本公开范围的限制。本领域技术人员可以根据需要，增添或替换其他部件。

计算设备1000可以是任何类型的静止或移动计算设备，包括移动计算机或移动计算设备（例如，平板计算机、个人数字助理、膝上型计算机、笔记本计算机、上网本等）、移动电话（例如，智能手机）、可佩戴的计算设备（例如，智能手表、智能眼镜等）或其他类型的移动设备，或者诸如台式计算机或个人计算机（PC，Personal Computer）的静止计算设备。计算设备1000还可以是移动式或静止式的服务器。

其中，处理器1020用于执行如下计算机程序/指令，该计算机程序/指令被处理器执行时实现上述模型训练方法或者代码处理方法的步骤。

本公开中的各个实施例均采用递进的方式描述，各个实施例之间相同相似的部分互相参见即可，每个实施例重点说明的都是与其他实施例的不同之处。尤其，对于计算设备实施例而言，由于其基本相似于模型训练方法或者代码处理方法实施例，所以描述的比较简单，相关之处参见模型训练方法或者代码处理方法实施例的部分说明即可。

本公开一实施例还提供一种计算机可读存储介质，其存储有计算机程序/指令，该计算机程序/指令被处理器执行时实现上述模型训练方法或者代码处理方法的步骤。

本公开中的各个实施例均采用递进的方式描述，各个实施例之间相同相似的部分互相参见即可，每个实施例重点说明的都是与其他实施例的不同之处。尤其，对于计算机可读存储介质实施例而言，由于其基本相似于模型训练方法或者代码处理方法实施例，所以描述的比较简单，相关之处参见模型训练方法或者代码处理方法实施例的部分说明即可。

本公开一实施例还提供一种计算机程序产品，包括计算机程序/指令，该计算机程序/指令被处理器执行时实现上述模型训练方法或者代码处理方法的步骤。

上述为本实施例的一种计算机程序产品的示意性方案。需要说明的是，该计算机程序产品的技术方案与上述的模型训练方法或者代码处理方法的技术方案属于同一构思，计算机程序产品的技术方案未详细描述的细节内容，均可以参见上述模型训练方法或者代码处理方法的技术方案的描述。

上述对本公开特定实施例进行了描述。其它实施例在所附权利要求书的范围内。在一些情况下，在权利要求书中记载的动作或步骤可以按照不同于实施例中的顺序来执行并且仍然可以实现期望的结果。另外，在附图中描绘的过程不一定要求示出的特定顺序或者连续顺序才能实现期望的结果。在某些实施方式中，多任务处理和并行处理也是可以的或者可能是有利的。

所述计算机指令包括计算机程序代码，所述计算机程序代码可以为源代码形式、对象

代码形式、可执行文件或某些中间形式等。所述计算机可读介质可以包括：能够携带所述计算机程序代码的任何实体或装置、记录介质、U 盘、移动硬盘、磁碟、光盘、计算机存储器、只读存储器(ROM,Read-Only Memory)、随机存取存储器(RAM,Random Access Memory)、电载波信号、电信信号以及软件分发介质等。需要说明的是，所述计算机可读介质包含的内容可以根据专利实践的要求进行适当的增减，例如在某些地区，根据专利实践，计算机可读介质不包括电载波信号和电信信号。

需要说明的是，对于前述的各方法实施例，为了简便描述，故将其都表述为一系列的动作组合，但是本领域技术人员应该知悉，本公开实施例并不受所描述的动作顺序的限制，因为依据本公开实施例，某些步骤可以采用其它顺序或者同时进行。其次，本领域技术人员也应该知悉，说明书中所描述的实施例均属于优选实施例，所涉及的动作和模块并不一定都是本公开实施例所必须的。

在上述实施例中，对各个实施例的描述都各有侧重，某个实施例中没有详述的部分，可以参见其它实施例的相关描述。

以上公开的本公开优选实施例只是用于帮助阐述本公开。可选实施例并没有详尽叙述所有的细节，也不限制该发明仅为所述的具体实施方式。显然，根据本公开实施例的内容，可作很多的修改和变化。本公开选取并具体描述这些实施例，是为了更好地解释本公开实施例的原理和实际应用，从而使所属技术领域技术人员能很好地理解和利用本公开。本公开仅受权利要求书及其全部范围和等效物的限制。

权 利 要 求 书

1. 一种模型训练方法，包括：

确定待训练代码处理模型对应的第一样本代码文件以及第二样本代码文件，其中，所述第一样本代码文件与所述第二样本代码文件存在调用关系；

基于所述第一样本代码文件，确定第一样本代码数据以及样本代码标签；

基于所述第二样本代码文件确定第二样本代码数据，并基于所述第一样本代码数据以及所述第二样本代码数据，确定样本代码数据；

基于所述样本代码数据以及所述样本代码标签，对所述待训练代码处理模型进行模型训练，获得训练完成的代码处理模型。

2. 根据权利要求 1 所述的模型训练方法，还包括：

确定待处理代码数据；

将所述待处理代码数据输入所述训练完成的代码处理模型，利用所述训练完成的代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作。

3. 根据权利要求 1 或 2 所述的模型训练方法，所述基于所述第一样本代码文件，确定第一样本代码数据以及样本代码标签，包括：

从所述第一样本代码文件中，确定调用所述第二样本代码文件的调用代码语句，并将所述调用代码语句，确定为所述样本代码标签；

从所述第一样本代码文件中，确定除所述样本代码标签之外的其他代码语句，并将所述其他代码语句确定为所述第一样本代码数据。

4. 根据权利要求 1 至 3 任一项所述的模型训练方法，所述基于所述第二样本代码文件确定第二样本代码数据，包括：

从所述第二样本代码文件中，确定初始调用代码数据；

对所述初始调用代码数据进行数据筛选处理，获得目标调用代码数据，并将所述目标调用代码数据，作为所述第二样本代码数据。

5. 根据权利要求 4 所述的模型训练方法，所述从所述第二样本代码文件中，确定初始调用代码数据，包括：

确定所述第二样本代码文件中的代码数据，并对所述代码数据进行语法分析，获得所述代码数据中的多个调用代码数据；

将各调用代码数据进行数据拼接，获得所述初始调用代码数据。

6. 根据权利要求 4 或 5 所述的模型训练方法，所述对所述初始调用代码数据进行数据筛选处理，获得目标调用代码数据，包括：

从所述初始调用代码数据中，确定第一调用代码数据、第二调用代码数据以及第三调用代码数据，其中，所述第一调用代码数据为被所述第一样本代码文件调用的数据，所述第二调用代码数据为被导入所述第二样本代码文件中的数据，所述第三调用代码数据为被全局定义的数据；

确定所述第一调用代码数据的第一数据数量；

基于所述第一数据数量以及预设数量比例，从所述第二调用代码数据以及所述第三调用代码数据中，选择与所述第一调用代码数据对应的第四调用代码数据；

将所述第一调用代码数据以及所述第四调用代码数据，确定为所述目标调用代码数据。

7. 根据权利要求 1 至 6 任一项所述的模型训练方法，所述确定待训练代码处理模型对应的第一样本代码文件以及第二样本代码文件，包括：

确定待训练代码处理模型对应的多个样本代码文件，并对各样本代码文件进行语法分析，获得各样本代码文件中的调用代码数据；

基于所述调用代码数据，从所述各样本代码文件中确定所述第一样本代码文件以及与所述第一样本代码文件存在调用关系的所述第二样本代码文件。

8. 根据权利要求 1 至 7 任一项所述的模型训练方法，所述基于所述样本代码数据以及所述样本代码标签，对所述待训练代码处理模型进行模型训练，获得训练完成的所述代码处理模型，包括：

利用训练数据模板对所述样本代码数据以及所述样本代码标签进行处理，获得模型训练数据；

基于所述模型训练数据对所述待训练代码处理模型进行模型训练，获得所述训练完成的代码处理模型。

9. 一种代码处理方法，包括：

接收用户通过客户端中的代码处理界面发送的待处理代码数据；

将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根据上述权利要求 1 至 8 任意一项所述的模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作；

将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示。

10. 根据权利要求 9 所述的代码处理方法，所述待处理代码数据为待补全代码数据；

所述将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，包括：

将所述待补全代码数据输入所述代码处理模型，利用所述代码处理模型对所述待补全代码数据进行所述代码补全操作，获得所述待补全代码数据对应的代码补全数据；

所述将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示，包括：

将所述代码补全数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述代码补全数据向所述用户展示。

11. 根据权利要求 9 所述的代码处理方法，所述待处理代码数据为待修复代码数据；

所述将所述待处理代码数据输入代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，包括：

将所述待修复代码数据输入所述代码处理模型，利用所述代码处理模型对所述待修复代码数据进行所述代码修复操作，获得所述待修复代码数据对应的代码修复数据；

所述将所述目标代码数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述目标代码数据向所述用户展示，包括：

将所述代码修复数据发送至所述客户端，使所述客户端通过所述代码处理界面将所述代码修复数据向所述用户展示。

12. 一种代码处理平台，包括请求接收单元、代码处理单元和响应单元；

所述请求接收单元，用于接收代码处理请求，并基于所述代码处理请求确定待处理代码数据以及所述待处理代码数据对应的代码处理模型标识；

所述代码处理单元，用于基于所述代码处理模型标识确定代码处理模型，将所述待处理代码数据输入所述代码处理模型，利用所述代码处理模型对所述待处理代码数据进行数据调整操作，获得所述待处理代码数据对应的目标代码数据，其中，所述代码处理模型根

据上述权利要求 1 至 8 任意一项所述的模型训练方法训练获得，所述数据调整操作包括代码补全操作、代码生成操作或者代码修复操作；

所述响应单元，用于输出所述目标代码数据。

13. 根据权利要求 12 所述的代码处理平台，所述请求接收单元，用于从所述代码处理请求中，确定所述待处理代码数据以及所述待处理代码数据对应的所述代码处理模型标识；或者

所述请求接收单元，用于确定所述代码处理请求中的代码生成文本以及所述代码生成文本对应的处理提示信息，基于所述处理提示信息将所述代码生成文本输入代码生成模型，获得所述待处理代码数据，并确定所述待处理代码数据对应的所述代码处理模型标识。

14. 一种计算设备，包括：

存储器和处理器；

所述存储器用于存储计算机程序/指令，所述处理器用于执行所述计算机程序/指令，该计算机程序/指令被处理器执行时实现权利要求 1 至 8 任意一项所述模型训练方法、权利要求 9 至 11 任意一项所述代码处理方法的步骤。

15. 一种计算机可读存储介质，其存储有计算机程序/指令，该计算机程序/指令被处理器执行时实现权利要求 1 至 8 任意一项所述模型训练方法、权利要求 9 至 11 任意一项所述代码处理方法的步骤。

16. 一种计算机程序产品，包括计算机程序/指令，该计算机程序/指令被处理器执行时实现权利要求 1 至 8 任意一项所述模型训练方法、权利要求 9 至 11 任意一项所述代码处理方法的步骤。