United States Patent Application Publication 20250265047
Kind Code A1
Publication Date August 21, 2025
Inventor(s) Wang; Chen et al.

ENERGY CONSUMPTION AWARE CREATION OF SOFTWARE CODE

Abstract

Computer-implemented methods for energy consumption aware automated creation of software code are provided. Aspects include receiving one or more commands from a user, wherein the one or more commands are programming code provided in a natural language and creating a plurality of playbook prompts based on the one or more commands. Aspects also include calculating a sustainability score for each of the plurality of playbook prompts and offering one or more of the plurality of playbook prompts along with the sustainability score for the user to select.

Inventors: **Wang; Chen (Chappaqua, NY), Chen; Huamin (Newton, MA)**

Applicant: **International Business Machines Corporation** (Armonk, NY)

Family ID: **1000007902030**

Appl. No.: **18/582715**

Filed: **February 21, 2024**

Publication Classification

Int. Cl.: **G06F8/35** (20180101); **G06F8/34** (20180101)

U.S. Cl.:

CPC **G06F8/35** (20130101); **G06F8/34** (20130101);

Background/Summary

BACKGROUND
[0001] The present disclosure generally relates to the automated creation of software code, and

more specifically, to energy consumption aware automated creation of software code.
[0002] Automated code generation refers to a process where software code is automatically generated from a high-level specification or model. In automated code generation systems, developers write code in a high-level language or use graphical tools to specify the desired functionality of a software system and a code generation tool automatically creates the software code that implements the functionality.
[0003] Open-source software developer communities, like Ansible, have created powerful automation tools for managing hybrid-cloud environments using a single, consistent automation platform. Ansible uses a human-readable programming language called YAML to create automation processes that are stored as Ansible Playbooks. A code generation model can be used to build automation for hybrid-cloud environments using natural language input. For example, by entering a coding command in a natural language, such as “Deploy Web Application Stack” or “Install Nodejs dependencies,” the code generation model parses the sentence and builds the requested automation workflow, which can be delivered as an Ansible Playbook. The generated playbook can be accepted as is or customized by the developer. As the demand for information technology (IT) software skills continues to rise, IT automation is crucial for scaling digital transformations in businesses.

SUMMARY

[0004] Embodiments of the present disclosure are directed to computer-implemented methods for energy consumption aware automated creation of software code. According to an aspect, a computer-implemented method includes receiving one or more commands from a user, wherein the one or more commands are programming code provided in a natural language and creating a plurality of playbook prompts based on the one or more commands. The method also includes calculating a sustainability score for each of the plurality of playbook prompts and offering one or more of the plurality of playbook prompts along with the sustainability score for the user to select.
[0005] Embodiments also include computing systems and computer program products for energy consumption aware automated creation of software code.
[0006] Additional technical features and benefits are realized through the techniques of the present disclosure. Embodiments and aspects of the disclosure are described in detail herein and are considered a part of the claimed subject matter. For a better understanding, refer to the detailed description and to the drawings.

Description

BRIEF DESCRIPTION OF THE DRAWINGS

[0007] The specifics of the exclusive rights described herein are particularly pointed out and distinctly claimed in the claims at the conclusion of the specification. The foregoing and other features and advantages of the embodiments of the present disclosure are apparent from the following detailed description taken in conjunction with the accompanying drawings in which:
[0008] FIG. **1** depicts a block diagram of an example computer system for use in conjunction with one or more embodiments of the present disclosure;
[0009] FIG. **2** depicts a block diagram of components of a machine learning training and inference system in accordance with one or more embodiments of the present disclosure;
[0010] FIG. **3** depicts a flowchart of a method for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure;
[0011] FIG. **4** depicts a flowchart of a method for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure;
[0012] FIG. **5** depicts a flowchart of a method for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure; and

[0013] FIG. **6** depicts a flowchart of a method for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure.

DETAILED DESCRIPTION

[0014] Existing methods for automated code generation are configured to create a computer-executable software code that implements a functionality that is specified by a user in a computer-executable software language that is specified by the user. The functionality can be provided by the user in the form of a user in a high-level programming language, a natural language, or via a series of inputs into graphical tools.

[0015] In general, multiple different versions of computer-executable software code can be created to implement a desired functionality. For example, if a group of software developers was given the task of manually creating computer-executable software code to achieve the same functionality, it would be expected that the computer-executable software code generated by each of the software developers would be unique, or at least different in several aspects. In addition, each of the versions of the software code developed by the developers would have different characteristics that may lead to different performance characteristics of the software code.

[0016] In most cases, automated code generation systems are only configured to output a single version of the computer-executable software code. As a result, the user of the automated code generation system is not presented with different versions of the computer-executable software code to choose from. In addition, even if an automated code generation system was configured to present the user with different versions of the code, the user may likely not be able to discern or appreciate the different characteristics of the versions of the code. As a result, users of automated code generation systems may often utilize computer-executable software that may not be optimal for performing the desired functionality.

[0017] In exemplary embodiments, systems, and methods for energy consumption aware automated creation of software code are provided. In exemplary embodiments, an automated code generation system is configured to receive a desired functionality from a user and to responsively generate a plurality of versions of computer-executable software code that perform the functionality. Each of the plurality of versions of computer-executable software code is then scored and presented to the user for selection along with the score calculated for the computer-executable software code. In one embodiment, the calculated score for the computer-executable software code is a sustainability score that evaluates the energy efficiency of executing the computer-executable software code. In another embodiment, the calculated score is a resource consumption score that evaluates the computing resources required to execute the computer-executable software code.

[0018] Various aspects of the present disclosure are described by narrative text, flowcharts, block diagrams of computer systems, and/or block diagrams of the machine logic included in computer program product (CPP) embodiments. With respect to any flowcharts, depending upon the technology involved, the operations can be performed in a different order than what is shown in a given flowchart. For example, again depending upon the technology involved, two operations shown in successive flowchart blocks may be performed in reverse order, as a single integrated step, concurrently, or in a manner at least partially overlapping in time.

[0019] A computer program product embodiment (“CPP embodiment” or “CPP”) is a term used in the present disclosure to describe any set of one, or more, storage media (also called “mediums”) collectively included in a set of one, or more, storage devices that collectively include machine readable code corresponding to instructions and/or data for performing computer operations specified in a given CPP claim. A “storage device” is any tangible device that can retain and store instructions for use by a computer processor. Without limitation, the computer-readable storage medium may be an electronic storage medium, a magnetic storage medium, an optical storage medium, an electromagnetic storage medium, a semiconductor storage medium, a mechanical storage medium, or any suitable combination of the foregoing. Some known types of storage devices that include these mediums include: diskette, hard disk, random access memory (RAM),

read-only memory (ROM), erasable programmable read-only memory (EPROM or Flash memory), static random access memory (SRAM), compact disc read-only memory (CD-ROM), digital versatile disk (DVD), memory stick, floppy disk, mechanically encoded device (such as punch cards or pits/lands formed in a major surface of a disc) or any suitable combination of the foregoing. A computer readable storage medium, as that term is used in the present disclosure, is not to be construed as storage in the form of transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide, light pulses passing through a fiber optic cable, electrical signals communicated through a wire, and/or other transmission media. As will be understood by those of skill in the art, data is typically moved at some occasional points in time during normal operations of a storage device, such as during access, de-fragmentation or garbage collection, but this does not render the storage device as transitory because the data is not transitory while it is stored.

[0020] Computing environment **100** contains an example of an environment for the execution of at least some of the computer code involved in performing the inventive methods, such as energy consumption aware automated creation of software code, as shown at block **150**. In addition to block **150**, computing environment **100** includes, for example, computer **101**, wide area network (WAN) **102**, end user device (EUD) **103**, remote server **104**, public Cloud **105**, and private Cloud **106**. In this embodiment, computer **101** includes processor set **110** (including processing circuitry **120** and cache **121**), communication fabric **111**, volatile memory **112**, persistent storage **113** (including operating system **122** and block **150**, as identified above), peripheral device set **114** (including user interface (UI), device set **123**, storage **124**, and Internet of Things (IoT) sensor set **125**), and network module **115**. Remote server **104** includes remote database **132**. Public Cloud **105** includes gateway **130**, Cloud orchestration module **131**, host physical machine set **142**, virtual machine set **143**, and container set **144**.

[0021] COMPUTER **101** may take the form of a desktop computer, laptop computer, tablet computer, smart phone, smart watch or other wearable computer, mainframe computer, quantum computer or any other form of computer or mobile device now known or to be developed in the future that is capable of running a program, accessing a network or querying a database, such as remote database **132**. As is well understood in the art of computer technology, and depending upon the technology, performance of a computer-implemented method may be distributed among multiple computers and/or between multiple locations. On the other hand, in this presentation of computing environment **100**, detailed discussion is focused on a single computer, specifically computer **101**, to keep the presentation as simple as possible. Computer **101** may be located in a Cloud, even though it is not shown in a Cloud in FIG. **1**. On the other hand, computer **101** is not required to be in a Cloud except to any extent as may be affirmatively indicated.

[0022] PROCESSOR SET **110** includes one, or more, computer processors of any type now known or to be developed in the future. Processing circuitry **120** may be distributed over multiple packages, for example, multiple, coordinated integrated circuit chips. Processing circuitry **120** may implement multiple processor threads and/or multiple processor cores. Cache **121** is memory that is located in the processor chip package(s) and is typically used for data or code that should be available for rapid access by the threads or cores running on processor set **110**. Cache memories are typically organized into multiple levels depending upon relative proximity to the processing circuitry. Alternatively, some, or all, of the cache for the processor set may be located “off chip.” In some computing environments, processor set **110** may be designed for working with qubits and performing quantum computing.

[0023] Computer readable program instructions are typically loaded onto computer **101** to cause a series of operational steps to be performed by processor set **110** of computer **101** and thereby effect a computer-implemented method, such that the instructions thus executed will instantiate the methods specified in flowcharts and/or narrative descriptions of computer-implemented methods included in this document (collectively referred to as “the inventive methods”). These computer

readable program instructions are stored in various types of computer readable storage media, such as cache **121** and the other storage media discussed below. The program instructions, and associated data, are accessed by processor set **110** to control and direct performance of the inventive methods. In computing environment **100**, at least some of the instructions for performing the inventive methods may be stored in block **150** in persistent storage **113**.

[0024] COMMUNICATION FABRIC **111** is the signal conduction paths that allow the various components of computer **101** to communicate with each other. Typically, this fabric is made of switches and electrically conductive paths, such as the switches and electrically conductive paths that make up busses, bridges, physical input/output ports and the like. Other types of signal communication paths may be used, such as fiber optic communication paths and/or wireless communication paths.

[0025] VOLATILE MEMORY **112** is any type of volatile memory now known or to be developed in the future. Examples include dynamic type random access memory (RAM) or static type RAM. Typically, the volatile memory is characterized by random access, but this is not required unless affirmatively indicated. In computer **101**, the volatile memory **112** is located in a single package and is internal to computer **101**, but, alternatively or additionally, the volatile memory may be distributed over multiple packages and/or located externally with respect to computer **101**.

[0026] PERSISTENT STORAGE **113** is any form of non-volatile storage for computers that is now known or to be developed in the future. The non-volatility of this storage means that the stored data is maintained regardless of whether power is being supplied to computer **101** and/or directly to persistent storage **113**. Persistent storage **113** may be a read only memory (ROM), but typically at least a portion of the persistent storage allows writing of data, deletion of data and re-writing of data. Some familiar forms of persistent storage include magnetic disks and solid state storage devices. Operating system **122** may take several forms, such as various known proprietary operating systems or open source Portable Operating System Interface type operating systems that employ a kernel. The code included in block **150** typically includes at least some of the computer code involved in performing the inventive methods.

[0027] PERIPHERAL DEVICE SET **114** includes the set of peripheral devices of computer **101**. Data communication connections between the peripheral devices and the other components of computer **101** may be implemented in various ways, such as Bluetooth connections, Near-Field Communication (NFC) connections, connections made by cables (such as universal serial bus (USB) type cables), insertion type connections (for example, secure digital (SD) card), connections made though local area communication networks and even connections made through wide area networks such as the internet. In various embodiments, UI device set **123** may include components such as a display screen, speaker, microphone, wearable devices (such as goggles and smart watches), keyboard, mouse, printer, touchpad, game controllers, and haptic devices. Storage **124** is external storage, such as an external hard drive, or insertable storage, such as an SD card. Storage **124** may be persistent and/or volatile. In some embodiments, storage **124** may take the form of a quantum computing storage device for storing data in the form of qubits. In embodiments where computer **101** is required to have a large amount of storage (for example, where computer **101** locally stores and manages a large database) then this storage may be provided by peripheral storage devices designed for storing very large amounts of data, such as a storage area network (SAN) that is shared by multiple, geographically distributed computers. IoT sensor set **125** is made up of sensors that can be used in Internet of Things applications. For example, one sensor may be a thermometer and another sensor may be a motion detector.

[0028] NETWORK MODULE **115** is the collection of computer software, hardware, and firmware that allows computer **101** to communicate with other computers through WAN **102**. Network module **115** may include hardware, such as modems or Wi-Fi signal transceivers, software for packetizing and/or de-packetizing data for communication network transmission, and/or web browser software for communicating data over the internet. In some embodiments, network control

functions and network forwarding functions of network module **115** are performed on the same physical hardware device. In other embodiments (for example, embodiments that utilize software-defined networking (SDN)), the control functions and the forwarding functions of network module **115** are performed on physically separate devices, such that the control functions manage several different network hardware devices. Computer readable program instructions for performing the inventive methods can typically be downloaded to computer **101** from an external computer or external storage device through a network adapter card or network interface included in network module **115**.

[0029] WAN **102** is any wide area network (for example, the internet) capable of communicating computer data over non-local distances by any technology for communicating computer data, now known or to be developed in the future. In some embodiments, the WAN may be replaced and/or supplemented by local area networks (LANs) designed to communicate data between devices located in a local area, such as a Wi-Fi network. The WAN and/or LANs typically include computer hardware such as copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway computers and edge servers.

[0030] END USER DEVICE (EUD) **103** is any computer system that is used and controlled by an end user (for example, a customer of an enterprise that operates computer **101**), and may take any of the forms discussed above in connection with computer **101**. EUD **103** typically receives helpful and useful data from the operations of computer **101**. For example, in a hypothetical case where computer **101** is designed to provide a recommendation to an end user, this recommendation would typically be communicated from network module **115** of computer **101** through WAN **102** to EUD **103**. In this way, EUD **103** can display, or otherwise present, the recommendation to an end user. In some embodiments, EUD **103** may be a client device, such as thin client, heavy client, mainframe computer, desktop computer and so on.

[0031] REMOTE SERVER **104** is any computer system that serves at least some data and/or functionality to computer **101**. Remote server **104** may be controlled and used by the same entity that operates computer **101**. Remote server **104** represents the machine(s) that collects and store helpful and useful data for use by other computers, such as computer **101**. For example, in a hypothetical case where computer **101** is designed and programmed to provide a recommendation based on historical data, then this historical data may be provided to computer **101** from remote database **132** of remote server **104**.

[0032] PUBLIC CLOUD **105** is any computer system available for use by multiple entities that provides on-demand availability of computer system resources and/or other computer capabilities, especially data storage (Cloud storage) and computing power, without direct active management by the user. Cloud computing typically leverages the sharing of resources to achieve coherence and economies of scale. The direct and active management of the computing resources of public Cloud **105** is performed by the computer hardware and/or software of Cloud orchestration module **131**. The computing resources provided by public Cloud **105** are typically implemented by virtual computing environments that run on various computers making up the computers of host physical machine set **142**, which is the universe of physical computers in and/or available to public Cloud **105**. The virtual computing environments (VCEs) typically take the form of virtual machines from virtual machine set **143** and/or containers from container set **144**. It is understood that these VCEs may be stored as images and may be transferred among and between the various physical machine hosts, either as images or after the instantiation of the VCE. Cloud orchestration module **131** manages the transfer and storage of images, deploys new instantiations of VCEs, and manages active instantiations of VCE deployments. Gateway **130** is the collection of computer software, hardware, and firmware that allows public Cloud **105** to communicate through WAN **102**.

[0033] Some further explanation of virtualized computing environments (VCEs) will now be provided. VCEs can be stored as “images.” A new active instance of the VCE can be instantiated from the image. Two familiar types of VCEs are virtual machines and containers. A container is a

VCE that uses operating-system-level virtualization. This refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances, called containers. These isolated user-space instances typically behave as real computers from the point of view of programs running in them. A computer program running on an ordinary operating system can utilize all resources of that computer, such as connected devices, files and folders, network shares, CPU power, and quantifiable hardware capabilities. However, programs running inside a container can only use the contents of the container and devices assigned to the container, a feature which is known as containerization.

[0034] PRIVATE CLOUD **106** is similar to public Cloud **105**, except that the computing resources are only available for use by a single enterprise. While private Cloud **106** is depicted as being in communication with WAN **102**, in other embodiments a private Cloud may be disconnected from the internet entirely and only accessible through a local/private network. A hybrid Cloud is a composition of multiple Clouds of different types (for example, private, community, or public Cloud types), often respectively implemented by different vendors. Each of the multiple Clouds remains a separate and discrete entity, but the larger hybrid Cloud architecture is bound together by standardized or proprietary technology that enables orchestration, management, and/or data/application portability between the multiple constituent Clouds. In this embodiment, public Cloud **105** and private Cloud **106** are both part of a larger hybrid Cloud.

[0035] One or more embodiments described herein can utilize machine learning techniques to perform prediction and or classification tasks, for example. In one or more embodiments, machine learning functionality can be implemented using an artificial neural network (ANN) having the capability to be trained to perform a function. In machine learning and cognitive science, ANNs are a family of statistical learning models inspired by the biological neural networks of animals, and in particular the brain. ANNs can be used to estimate or approximate systems and functions that depend on a large number of inputs. Convolutional neural networks (CNN) are a class of deep, feed-forward ANNs that are particularly useful for tasks such as, but not limited to analyzing visual imagery and natural language processing (NLP). Recurrent neural networks (RNN) are another class of deep, feed-forward ANNs and are particularly useful at tasks such as, but not limited to, unsegmented connected handwriting recognition and speech recognition. Other types of neural networks are also known and can be used in accordance with one or more embodiments described herein.

[0036] ANNs can be embodied as so-called “neuromorphic” systems of interconnected processor elements that act as simulated “neurons” and exchange “messages” between each other in the form of electronic signals. Similar to the so-called “plasticity” of synaptic neurotransmitter connections that carry messages between biological neurons, the connections in ANNs that carry electronic messages between simulated neurons are provided with numeric weights that correspond to the strength or weakness of a given connection. The weights can be adjusted and tuned based on experience, making ANNs adaptive to inputs and capable of learning. For example, an ANN for handwriting recognition is defined by a set of input neurons that can be activated by the pixels of an input image. After being weighted and transformed by a function determined by the network's designer, the activation of these input neurons are then passed to other downstream neurons, which are often referred to as “hidden” neurons. This process is repeated until an output neuron is activated. The activated output neuron determines which character was input.

[0037] A container is a VCE that uses operating-system-level virtualization. This refers to an operating system feature in which the kernel allows the existence of multiple isolated user-space instances, called containers. These isolated user-space instances typically behave as real computers from the point of view of programs running in them. A computer program running on an ordinary operating system can utilize all resources of that computer, such as connected devices, files and folders, network shares, CPU power, and quantifiable hardware capabilities. However, programs running inside a container can only use the contents of the container and devices assigned to the

container, a feature which is known as containerization.
[0038] Systems for training and using a machine learning model are now described in more detail with reference to FIG. **2**. Particularly, FIG. **2** depicts a block diagram of components of a machine learning training and inference system **200** according to one or more embodiments described herein. The system **200** performs training **202** and inference **204**. During training **202**, a training engine **216** trains a model (e.g., the trained model **218**) to perform a task, such as scoring a sustainability or energy consumption associated with a computer executable software code or playbook. Inference **204** is the process of implementing the trained model **218** to perform the task, such as scoring a sustainability or energy consumption associated with a computer executable software code or playbook, in the context of a larger system (e.g., a system **226**). All or a portion of the system **200** shown in FIG. **2** can be implemented, for example by all or a subset of the computing environment **100** of FIG. **1**.
[0039] The training **202** begins with training data **212**, which may be structured or unstructured data. According to one or more embodiments described herein, the training data **212** includes computer-executable software code or playbook and user-provided sustainability or energy consumption scores associated with each computer-executable software code or playbook. The training engine **216** receives the training data **212** and a model form **214**. The model form **214** represents an untrained base model. The model form **214** can have preset weights and biases, which can be adjusted during training. It should be appreciated that the model form **214** can be selected from many different model forms depending on the task to be performed. For example, where the training **202** is to train a model to perform image classification, the model form **214** may be a model form of a CNN. The training **202** can be supervised learning, semi-supervised learning, unsupervised learning, reinforcement learning, and/or the like, including combinations and/or multiples thereof. For example, supervised learning can be used to train a machine learning model to classify an object of interest in an image. To do this, the training data **212** includes labeled images, including images of the object of interest with associated labels (ground truth) and other images that do not include the object of interest with associated labels. In this example, the training engine **216** takes as input a training image from the training data **212**, makes a prediction for classifying the image, and compares the prediction to the known label. The training engine **216** then adjusts weights and/or biases of the model based on the results of the comparison, such as by using backpropagation. The training **202** may be performed multiple times (referred to as “epochs”) until a suitable model is trained (e.g., the trained model **218**).
[0040] Once trained, the trained model **218** can be used to perform inference **204** to perform a task, such as scoring a sustainability or energy consumption associated with a computer executable software code or playbook. The inference engine **220** applies the trained model **218** to new data **222** (e.g., real-world, non-training data). For example, if the trained model **218** is trained to classify images of a particular object, such as a chair, the new data **222** can be an image of a chair that was not part of the training data **212**. In this way, the new data **222** represents data to which the model **218** has not been exposed. The inference engine **220** makes a prediction **224** (e.g., a classification of an object in an image of the new data **222**) and passes the prediction **224** to the system **226**. The system **226** can, based on the prediction **224**, take an action, perform an operation, perform an analysis, and/or the like, including combinations and/or multiples thereof. In some embodiments, the system **226** can add to and/or modify the new data **222** based on the prediction **224**.
[0041] In accordance with one or more embodiments, the predictions **224** generated by the inference engine **220** are periodically monitored and verified to ensure that the inference engine **220** is operating as expected. Based on the verification, additional training **202** may occur using the trained model **218** as the starting point. The additional training **202** may include all or a subset of the original training data **212** and/or new training data **212**. In accordance with one or more embodiments, the training **202** includes updating the trained model **218** to account for changes in expected input data.

[0042] In exemplary embodiments, a code generation tool is configured to receive a natural language input, such as human-readable programming language called YAML, that specifies a desired functionality. The code generation tool is further configured to responsively generate multiple versions of computer-executable software code that each implement the functionality, such as an Ansible Playbooks. The generated playbooks are then each scored and one or more of the scored playbooks are presented to a user along with the score, or an indication of the score.
[0043] Referring now to FIG. **3**, a flowchart of a method **300** for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure is shown. As shown at block **302**, the method **300** includes receiving one or more commands from a user, where the one or more commands are programming code provided in a natural language. Next, as shown at block **304**, the method **300** includes creating a plurality of playbook prompts based on the one or more commands. In exemplary embodiments, each of the plurality of playbook prompts are computer-executable programming code that, when executed, will cause a processing system to perform the one or more commands received from the user. In one embodiment, the plurality of playbook prompts are Ansible playbooks, wherein Ansible is a human-readable programming language.
[0044] Next, as shown at block **306**, the method **300** includes calculating a sustainability score for each of the plurality of playbook prompts. In one embodiment, the sustainability score for each of the plurality of playbook prompts is calculated based on one or more keywords, wherein the one or more keywords is found from searching through different services, states, metadata, applications, and versions of applications utilized by the corresponding playbook prompts. In another embodiment, the sustainability score for each of the plurality of playbook prompts is calculated by inputting each of the plurality of playbook prompts into a trained prediction model, where the trained prediction model was trained using historical playbook prompts and sustainability scores assigned to each of the using historical playbook prompts.
[0045] Next, as shown at block **308**, the method **300** includes offering one or more of the plurality of playbook prompts along with the sustainability score for the user to select. In one embodiment, a sustainability level is assigned to each of the plurality of playbook prompts based on the sustainability score corresponding to each of the plurality of playbook prompts, where the sustainability level is selected from one of a high level of sustainability, a medium level of sustainability, and a low level of sustainability. In exemplary embodiments, the one or more of the plurality of playbook prompts are offered to the user in a color-coded manner, where a color of the one or more of the plurality of playbook prompts is selected based on the sustainability level.
[0046] Referring now to FIG. **4**, a flowchart of a method **400** for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure is shown. As shown at block **402**, the method **400** includes obtain a set of historical playbook prompts. Next, as shown at block **404**, the method **400** includes obtaining a sustainability score for each of the historical playbook prompts. In one embodiment, each of the historical playbook prompts has a corresponding sustainability score that was assigned by an individual, and the sustainability score reflects the energy efficiency of executing the playbook prompt.
[0047] Next, as shown at block **406**, the method **400** includes training a machine learning model based on the set of historical playbook prompts and the sustainability score for each of the historical playbook prompts, as described above in more detail with reference to FIG. **2**. The method **400** also includes receiving a new playbook prompt, (i.e., a playbook prompt that was not previously used to train the machine learning model), as shown at block **408**. At block **410**, the method **400** includes inputting the playbook prompt into the trained machine-learning model. The method **400** concludes at block **412** by receiving a sustainability score for the playbook prompt from the trained machine learning model.
[0048] In exemplary embodiments, a plurality of new playbook prompts that each are configured to perform the same functionality is provided to the trained machine learning model to be scored.

Once sustainability scores for each of the plurality of new playbook prompts are received, one or more of the plurality of new playbook prompts along with their corresponding sustainability scores are provided to a user. In one embodiment, the one or more of the plurality of new playbook prompts may include the playbook prompts having sustainability scores that exceed a threshold minimum value.

[0049] Referring now to FIG. **5**, a flowchart of a method for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure is shown. As shown at block **502**, the method **500** includes receiving a playbook prompt. In exemplary embodiments, the playbook prompt is automatically generated by a code generation tool based on a natural language input received from a user. Next, as shown at block **504**, the method **500** includes identifying a set of keywords corresponding to the playbook prompt. In an exemplary embodiment, the keywords identify known services, applications, states, and versions of applications that may be utilized by playbook prompts. In exemplary embodiments, the keywords are found by searching through different services, states, metadata, applications, and versions of applications utilized by the received playbook prompt. Next, as shown at block **506**, the method **500** includes identifying a sustainability impact corresponding to each of the set of keywords. In exemplary embodiments, the sustainability impact of each of the set of keywords is a sustainability impact score that indicates the sustainability of the service, state, or application associated with the keyword. In one embodiment, the sustainability impact score for each of the keywords is set by an individual. Next, at block **508**, the method **500** includes calculating a sustainability score for the playbook prompt based on the sustainability impacts of the set of keywords. In one embodiment, the sustainability score for the playbook prompt is the sum of the sustainability impact scores for the identified keywords associated with the playbook prompt. In another embodiment, the sustainability score for the playbook prompt is the average of the sustainability impact scores for the identified keywords associated with the playbook prompt.

[0050] In exemplary embodiments, a plurality of new playbook prompts that each are configured to perform the same functionality are scored based on their associated keywords. Once sustainability scores for each of the plurality of new playbook prompts are calculated, one or more of the plurality of new playbook prompts along with their corresponding sustainability scores are provided to a user. In one embodiment, the one or more of the plurality of new playbook prompts may include the playbook prompts having sustainability scores that exceed a threshold minimum value.

[0051] Referring now to FIG. **6**, a flowchart of a method **600** for energy consumption aware automated creation of software code in accordance with one or more embodiments of the present disclosure. As shown at block **602**, the method **600** includes obtaining a set of historical computer-executable software code. Next, as shown at block **604**, the method **600** includes obtaining a score for each of the historical computer-executable software code. In exemplary embodiments, the score is one of a sustainability score and energy consumption score that was assigned to the computer-executable software code by an individual. Next, at block **606**, the method **600** includes training a machine learning model based on the set of historical computer-executable software code and the score for each of the historical computer-executable software code, as explained in more detail above with reference to FIG. **2**.

[0052] Next, as shown at block **608**, the method **600** includes receiving a plurality of versions of a new computer-executable software code (i.e., computer-executable software code that was not used to train the machine learning model). In exemplary embodiments, the plurality of versions of a new computer-executable software code are automatically generated by a code generation tool based on a natural language input received from a user. The method **600** also includes inputting the plurality of versions of a new computer-executable software code into the trained machine-learning model, as shown at block **610**. Next, at block **612**, the method **600** includes receiving a score for the new computer-executable software code from the trained machine-learning model. In exemplary

embodiments, the score is one of a sustainability score and an energy consumption score.
[0053] The method **600** also includes presenting the new computer-executable software code and the score to a user, as shown at block **614**. The method **600** may also include receiving a selection of one of the plurality of versions of a new computer-executable software code from the user and deploying the selected new computer-executable software code in a computing system.
[0054] In exemplary embodiments, by generating multiple versions of the computer-executable software code, scoring the sustainability and/or energy consumption of the versions of the computer-executable software code, and presenting the user with the scored computer-executable software code for selection, the sustainability and energy efficiency of a computer executing the computer-executable software code is improved.
[0055] Various embodiments are described herein with reference to the related drawings. Alternative embodiments can be devised without departing from the scope of the present disclosure. Various connections and positional relationships (e.g., over, below, adjacent, etc.) are set forth between elements in the following description and in the drawings. These connections and/or positional relationships, unless specified otherwise, can be direct or indirect, and the present disclosure is not intended to be limiting in this respect. Accordingly, a coupling of entities can refer to either a direct or an indirect coupling, and a positional relationship between entities can be a direct or indirect positional relationship. Moreover, the various tasks and process steps described herein can be incorporated into a more comprehensive procedure or process having additional steps or functionality not described in detail herein.
[0056] One or more of the methods described herein can be implemented with any or a combination of the following technologies, which are each well known in the art: a discrete logic circuit(s) having logic gates for implementing logic functions upon data signals, an application specific integrated circuit (ASIC) having appropriate combinational logic gates, a programmable gate array(s) (PGA), a field programmable gate array (FPGA), etc.
[0057] For the sake of brevity, conventional techniques related to making and using aspects of the present disclosure may or may not be described in detail herein. In particular, various aspects of computing systems and specific computer programs to implement the various technical features described herein are well known. Accordingly, in the interest of brevity, many conventional implementation details are only mentioned briefly herein or are omitted entirely without providing the well-known system and/or process details.
[0058] In some embodiments, various functions or acts can take place at a given location and/or in connection with the operation of one or more apparatuses or systems. In some embodiments, a portion of a given function or act can be performed at a first device or location, and the remainder of the function or act can be performed at one or more additional devices or locations.
[0059] The terminology used herein is for the purpose of describing particular embodiments only and is not intended to be limiting. As used herein, the singular forms “a”, “an” and “the” are intended to include the plural forms as well, unless the context clearly indicates otherwise. It will be further understood that the terms “comprises” and/or “comprising,” when used in this specification, specify the presence of stated features, integers, steps, operations, elements, and/or components, but do not preclude the presence or addition of one or more other features, integers, steps, operations, element components, and/or groups thereof.
[0060] The corresponding structures, materials, acts, and equivalents of all means or step plus function elements in the claims below are intended to include any structure, material, or act for performing the function in combination with other claimed elements as specifically claimed. The present disclosure has been presented for purposes of illustration and description, but is not intended to be exhaustive or limited to the form disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the disclosure. The embodiments were chosen and described in order to best explain the principles of the disclosure and the practical application, and to enable others of ordinary skill in the art to

understand the disclosure for various embodiments with various modifications as are suited to the particular use contemplated.

[0061] The diagrams depicted herein are illustrative. There can be many variations to the diagram or the steps (or operations) described therein without departing from the spirit of the disclosure. For instance, the actions can be performed in a differing order or actions can be added, deleted or modified. Also, the term “coupled” describes having a signal path between two elements and does not imply a direct connection between the elements with no intervening elements/connections therebetween. All of these variations are considered a part of the present disclosure.

[0062] The following definitions and abbreviations are to be used for the interpretation of the claims and the specification. As used herein, the terms “comprises,” “comprising,” “includes,” “including,” “has,” “having,” “contains” or “containing,” or any other variation thereof, are intended to cover a non-exclusive inclusion. For example, a composition, a mixture, process, method, article, or apparatus that comprises a list of elements is not necessarily limited to only those elements but can include other elements not expressly listed or inherent to such composition, mixture, process, method, article, or apparatus.

[0063] Additionally, the term “exemplary” is used herein to mean “serving as an example, instance or illustration.” Any embodiment or design described herein as “exemplary” is not necessarily to be construed as preferred or advantageous over other embodiments or designs. The terms “at least one” and “one or more” are understood to include any integer number greater than or equal to one, i.e. one, two, three, four, etc. The terms “a plurality” are understood to include any integer number greater than or equal to two, i.e. two, three, four, five, etc. The term “connection” can include both an indirect “connection” and a direct “connection.”

[0064] The terms “about,” “substantially,” “approximately,” and variations thereof, are intended to include the degree of error associated with measurement of the particular quantity based upon the equipment available at the time of filing the application. For example, “about” can include a range of ±8% or 5%, or 2% of a given value.

[0065] The present disclosure may be a system, a method, and/or a computer program product at any possible technical detail level of integration. The computer program product may include a computer readable storage medium (or media) having computer readable program instructions thereon for causing a processor to carry out aspects of the present disclosure.

[0066] The computer readable storage medium can be a tangible device that can retain and store instructions for use by an instruction execution device. The computer readable storage medium may be, for example, but is not limited to, an electronic storage device, a magnetic storage device, an optical storage device, an electromagnetic storage device, a semiconductor storage device, or any suitable combination of the foregoing. A non-exhaustive list of more specific examples of the computer readable storage medium includes the following: a portable computer diskette, a hard disk, a random access memory (RAM), a read-only memory (ROM), an erasable programmable read-only memory (EPROM or Flash memory), a static random access memory (SRAM), a portable compact disc read-only memory (CD-ROM), a digital versatile disk (DVD), a memory stick, a floppy disk, a mechanically encoded device such as punch-cards or raised structures in a groove having instructions recorded thereon, and any suitable combination of the foregoing. A computer readable storage medium, as used herein, is not to be construed as being transitory signals per se, such as radio waves or other freely propagating electromagnetic waves, electromagnetic waves propagating through a waveguide or other transmission media (e.g., light pulses passing through a fiber-optic cable), or electrical signals transmitted through a wire.

[0067] Computer readable program instructions described herein can be downloaded to respective computing/processing devices from a computer readable storage medium or to an external computer or external storage device via a network, for example, the Internet, a local area network, a wide area network and/or a wireless network. The network may comprise copper transmission cables, optical transmission fibers, wireless transmission, routers, firewalls, switches, gateway

computers and/or edge servers. A network adapter card or network interface in each computing/processing device receives computer readable program instructions from the network and forwards the computer readable program instructions for storage in a computer readable storage medium within the respective computing/processing device.

[0068] Computer readable program instructions for carrying out operations of the present disclosure may be assembler instructions, instruction-set-architecture (ISA) instructions, machine instructions, machine dependent instructions, microcode, firmware instructions, state-setting data, configuration data for integrated circuitry, or either source code or object code written in any combination of one or more programming languages, including an object oriented programming language such as Smalltalk, C++, or the like, and procedural programming languages, such as the "C" programming language or similar programming languages. The computer readable program instructions may execute entirely on the user's computer, partly on the user's computer, as a stand-alone software package, partly on the user's computer and partly on a remote computer or entirely on the remote computer or server. In the latter scenario, the remote computer may be connected to the user's computer through any type of network, including a local area network (LAN) or a wide area network (WAN), or the connection may be made to an external computer (for example, through the Internet using an Internet Service Provider). In some embodiments, electronic circuitry including, for example, programmable logic circuitry, field-programmable gate arrays (FPGA), or programmable logic arrays (PLA) may execute the computer readable program instruction by utilizing state information of the computer readable program instructions to personalize the electronic circuitry, in order to perform aspects of the present disclosure.

[0069] Aspects of the present disclosure are described herein with reference to flowchart illustrations and/or block diagrams of methods, apparatus (systems), and computer program products according to embodiments of the present disclosure. It will be understood that each block of the flowchart illustrations and/or block diagrams, and combinations of blocks in the flowchart illustrations and/or block diagrams, can be implemented by computer readable program instructions.

[0070] These computer readable program instructions may be provided to a processor of a general purpose computer, special purpose computer, or other programmable data processing apparatus to produce a machine, such that the instructions, which execute via the processor of the computer or other programmable data processing apparatus, create means for implementing the functions/acts specified in the flowchart and/or block diagram block or blocks. These computer readable program instructions may also be stored in a computer readable storage medium that can direct a computer, a programmable data processing apparatus, and/or other devices to function in a particular manner, such that the computer readable storage medium having instructions stored therein comprises an article of manufacture including instructions which implement aspects of the function/act specified in the flowchart and/or block diagram block or blocks.

[0071] The computer readable program instructions may also be loaded onto a computer, other programmable data processing apparatus, or other device to cause a series of operational steps to be performed on the computer, other programmable apparatus or other device to produce a computer implemented process, such that the instructions which execute on the computer, other programmable apparatus, or other device implement the functions/acts specified in the flowchart and/or block diagram block or blocks.

[0072] The flowchart and block diagrams in the Figures illustrate the architecture, functionality, and operation of possible implementations of systems, methods, and computer program products according to various embodiments of the present disclosure. In this regard, each block in the flowchart or block diagrams may represent a module, segment, or portion of instructions, which comprises one or more executable instructions for implementing the specified logical function(s). In some alternative implementations, the functions noted in the blocks may occur out of the order noted in the Figures. For example, two blocks shown in succession may, in fact, be executed

substantially concurrently, or the blocks may sometimes be executed in the reverse order, depending upon the functionality involved. It will also be noted that each block of the block diagrams and/or flowchart illustration, and combinations of blocks in the block diagrams and/or flowchart illustration, can be implemented by special purpose hardware-based systems that perform the specified functions or acts or carry out combinations of special purpose hardware and computer instructions.

[0073] The descriptions of the various embodiments of the present disclosure have been presented for purposes of illustration, but are not intended to be exhaustive or limited to the embodiments disclosed. Many modifications and variations will be apparent to those of ordinary skill in the art without departing from the scope and spirit of the described embodiments. The terminology used herein was chosen to best explain the principles of the embodiments, the practical application or technical improvement over technologies found in the marketplace, or to enable others of ordinary skill in the art to understand the embodiments described herein.

Claims

1. A computer-implemented method for energy consumption aware automated creation of software code, the method comprising: receiving one or more commands from a user, wherein the one or more commands are programming code provided in a natural language; creating a plurality of playbook prompts based on the one or more commands; calculating a sustainability score for each of the plurality of playbook prompts; and offering one or more of the plurality of playbook prompts along with the sustainability score for the user to select.

2. The method of claim 1, wherein each of the plurality of playbook prompts are computer-executable programming code that, when executed, will cause a processing system to perform the one or more commands received from the user.

3. The method of claim 2, wherein the plurality of playbook prompts are Ansible playbooks, wherein Ansible is a human-readable programming language.

4. The method of claim 1, wherein the sustainability score for each of the plurality of playbook prompts is calculated based on one or more keywords, wherein the one or more keywords is found from searching through different services, states, metadata, applications, and versions of applications utilized by the corresponding playbook prompts.

5. The method of claim 1, wherein the sustainability score for each of the plurality of playbook prompts is calculated by inputting each of the plurality of playbook prompts into a trained prediction model.

6. The method of claim 5, wherein the trained prediction model was trained using historical playbook prompts and sustainability scores assigned to each of the using historical playbook prompts.

7. The method of claim 1, further comprising assigning a sustainability level to each of the plurality of playbook prompts based on the sustainability score corresponding to each of the plurality of playbook prompts, wherein the sustainability level is selected from one of a high level of sustainability, a medium level of sustainability, and a low level of sustainability.

8. The method of claim 7, wherein the one or more of the plurality of playbook prompts are offered to the user in a color-coded manner, where a color of the one or more of the plurality of playbook prompts is selected based on the sustainability level.

9. A computing system having a memory having computer readable instructions and one or more processors for executing the computer readable instructions, the computer readable instructions controlling the one or more processors to perform operations comprising: receiving one or more commands from a user, wherein the one or more commands are programming code provided in a natural language; creating a plurality of playbook prompts based on the one or more commands; calculating a sustainability score for each of the plurality of playbook prompts; and offering one or

more of the plurality of playbook prompts along with the sustainability score for the user to select.

10. The computing system of claim 9, wherein each of the plurality of playbook prompts are computer-executable programming code that, when executed, will cause a processing system to perform the one or more commands received from the user.

11. The computing system of claim 10, wherein the plurality of playbook prompts are Ansible playbooks, wherein Ansible is a human-readable programming language.

12. The computing system of claim 9, wherein the sustainability score for each of the plurality of playbook prompts is calculated based on one or more keywords, wherein the one or more keywords is found from searching through different services, states, metadata, applications, and versions of applications utilized by the corresponding playbook prompts.

13. The computing system of claim 9, wherein the sustainability score for each of the plurality of playbook prompts is calculated by inputting each of the plurality of playbook prompts into a trained prediction model.

14. The computing system of claim 13, wherein the trained prediction model was trained using historical playbook prompts and sustainability scores assigned to each of the using historical playbook prompts.

15. The computing system of claim 9, wherein the operations further comprise assigning a sustainability level to each of the plurality of playbook prompts based on the sustainability score corresponding to each of the plurality of playbook prompts, wherein the sustainability level is selected from one of a high level of sustainability, a medium level of sustainability, and a low level of sustainability.

16. The computing system of claim 15, wherein the one or more of the plurality of playbook prompts are offered to the user in a color-coded manner, where a color of the one or more of the plurality of playbook prompts is selected based on the sustainability level.

17. A computer program product comprising a computer readable storage medium having program instructions embodied therewith, the program instructions executable by a processor to cause the processor to perform operations comprising: receiving one or more commands from a user, wherein the one or more commands are programming code provided in a natural language; creating a plurality of playbook prompts based on the one or more commands; calculating a sustainability score for each of the plurality of playbook prompts; and offering one or more of the plurality of playbook prompts along with the sustainability score for the user to select.

18. The computer program product of claim 17, wherein each of the plurality of playbook prompts are computer-executable programming code that, when executed, will cause a processing system to perform the one or more commands received from the user.

19. The computer program product of claim 18, wherein the plurality of playbook prompts are Ansible playbooks, wherein Ansible is a human-readable programming language.

20. The computer program product of claim 17, wherein the sustainability score for each of the plurality of playbook prompts is calculated based on one or more keywords, wherein the one or more keywords is found from searching through different services, states, metadata, applications, and versions of applications utilized by the corresponding playbook prompts.