"AI Discovery"

Cross-Reference to Related Applications

[0001] The present application claims priority from Australian Provisional Patent Application No 2024900266 filed on 6 February 2024, the contents of which are incorporated herein by reference in their entirety.

Technical Field

[0002] This disclosure relates generally to detecting artificial intelligence (AI) implementation in a software application.

Background

[0003] In the recent surge of artificial intelligence (AI), more software applications are beginning to incorporate and utilise AI than ever before. In particular, more application packages (APs) such as application programming interfaces (APIs) and libraries are readily available for developers to use when developing their software applications. The advances in AI and successful deployment on devices, such as mobile devices, have enhanced the ability of software applications in daily activities, such as optical character recognition, autonomous driving and object recognition and shopping recommendations.

[0004] To enable AI capabilities for software applications and facilitate application development, several vendors released their AI frameworks to the public, such as TensorFlow Lite (TFLite) (https://www.tensorflow.org/), Caffe2 (https://github.com/pytorch/pytorch/tree/main/caffe2), Core ML (https://developer.apple.com/documentation/coreml) and ncnn (https://github.com/Tencent/ncnn). These frameworks aim to provide timely on-device model building and inference, as compared to the cloud-based models. On-device model inference also provides better protects user privacy, saves bandwidth load, and speeds up the inference time.

[0005] However, due to the surge of AI implementation in software applications, it can often be difficult to determine whether a software application does in fact utilise AI, especially given that new APs that use AI are continuously being developed and released. This makes it difficult

to catalogue all APs that utilise AI as the catalogue must be continuously updated, which can be very time consuming. To have a full scope of the current stage of AI services, identifying the characteristics of AI capability inside real-world applications is necessary. This process can be referred to as AI Discovery.

[0006] Detecting AI implementation in software applications can be important for numerous reasons. Most importantly, AI implementation in software applications is a cybersecurity issue. For example, a developer may use an AP that contains a malicious AI algorithm which may produce an undesired result for the developer and/or the user of the software application. This is especially important given the constant released of new APs that use AI, as the new APs may not have gained enough goodwill before being implemented into a software application. Further, as has been already seen, there will likely be more regulations on the use of AI due to inherent and unknown risks that the use of AI may produce. As such, detecting AI implementation aids in ensuring responsible use of AI technology.

[0007] Current methods of detecting AI implementation in software applications have a few limitations: (1) The existing focus is mainly on understanding the characteristics of on-device AI models, lacking the attention to other AP candidates such as package names, asset names, constant strings, and invoked APIs. Analysing the functionality of these candidates can provide a more comprehensive understanding of AI services; (2) Current approaches to studying on-device AI models are either based on manual inspection or pre-defined keyword searches. These ways are costly as they require up-to-date expertise in on-device AI. (3) Current automated AI Discovery techniques heavily rely on human-defined rule-based methods. These rules are limited by domain knowledge and hence infeasible to discover candidates in the wild.

[0008] In particular, these rule-based approaches identify patterns, such as popular AI APIs, model filename extensions, and specific keywords in binary libraries, to categorize apps as deep learning (DL) or AI-based. While effective, these methods are computationally inefficient, time-consuming, lack of adaptability to new AI models, and are unable to discern the domain of different employed AI technologies, such as Natural Language Processing (NLP) or Computer Vision (CV). There is a further disadvantage that a specialist, such as a senior software developer or machine learning scientist, is required to analyse the results of these rule-based approaches. In light of the limitations of these methods of detecting AI implementation, what is needed is a simple and computationally efficient method to detect AI implementation in software applications, where the results of the detect can be easily interpreted or understood.

[0009] Any discussion of documents, acts, materials, devices, articles or the like which has been included in the present specification is not to be taken as an admission that any or all of these matters form part of the prior art base or were common general knowledge in the field relevant to the present disclosure as it existed before the priority date of each of the appended claims.

[0010] Throughout this specification the word "comprise", or variations such as "comprises" or "comprising", will be understood to imply the inclusion of a stated element, integer or step, or group of elements, integers or steps, but not the exclusion of any other element, integer or step, or group of elements, integers or steps.

Summary

[0011] Disclosed herein are methods and systems for detecting artificial intelligence (AI) implementation in a software application. In particular, the methods disclosed herein are directed towards detecting AI implementation in a software application comprising one or more application packages (APs). In general, the disclosed methods are used to determine whether the APs of a software application are related to AI.

[0012] According to the present disclosure, there is provided a computer-implemented method for detecting artificial intelligence (AI) implementation in a software application comprising one or more application packages (APs). The method comprises:

extracting one or more application package (AP) strings from the software application, each of the one or more AP strings representing one of the one or more APs;

creating a prompt for a machine learning model, trained to generate output text, comprising the one or more AP strings, the prompt representing instructions to provide a classification of each of the one or more APs and provide functionality information of each of the one or more APs, the classification being AI relevant or non-AI relevant and the functionality information describing a functionality of the respective AP;

evaluating the machine learning model on the prompt to generate output text corresponding to the classification and the functionality information of each of the one or more APs; and

generating a report of the AI implementation in the software application based on the output text.

[0013] It is an advantage to create a prompt for a machine learning model comprising the one or more AP strings as rule-based methods of detecting AI implementation in a software application cannot accurately determine AI relevant APs. In particular, more software applications are beginning to utilise AI and new AI relevant APs are being developed. As a result, rule-based methods of detecting AI implementation cannot keep up with the constantly new AI relevant APs are being developed. The generative language model also provides an automated and efficient description of AI relevant APs.

[0014] In some embodiments, creating the prompt for the machine learning model comprises creating a first prompt and a second prompt. The first prompt represents instructions to provide the functionality information of each of the one or more APs. The second prompt represents instructions to provide the classification of each of the one or more APs.

[0015] In some embodiments, evaluating the machine learning model on the prompt comprises evaluating the machine learning model on the first prompt to generate initial output text and evaluating the machine learning model on the second prompt to generate the output text. The second prompt is based on the initial output text.

[0016] In some embodiments, the method further comprises analysing a description of the software application before extracting one or more AP strings from the software application to identify the one or more APs.

[0017] In some embodiments, analysing the description of the software application comprises performing a keyword search using terms relevant to AI.

[0018] In some embodiments, analysing a description of the software application comprises creating an initial prompt based on the description of the software application. The initial prompt represents instructions to provide a classification of the software application, the classification of the software application being likely to implement AI or unlikely to implement AI.

[0019] In some embodiments, extracting the one or more AP strings from the software application comprises applying a static analysis method to the software application.

[0020] In some embodiments, the method further comprises creating a third prompt based on the output text. The third prompt represents instructions to generate the report and generating the

report comprises evaluating the machine learning model on the third prompt to generate the report.

[0021] In some embodiments, the method further comprises classifying each of the one or more APs as an application programming interface (API) or a non-API and creating the third prompt further represents instructions to exclude APs classified as an API from the report.

[0022] In some embodiments, the first, second or third prompt comprises learning examples indicative of a desired output from evaluating the machine learning model on the first, second or third prompt, respectively.

[0023] In some embodiments, the learning examples are few-shot learning examples.

[0024] In some embodiments, the method further comprises storing each AP string of APs classified as AI relevant and respective functionality information on an AI relevant database and storing each AP string of APs classified as non-AI relevant and respective functionality information on a non-AI relevant database.

[0025] In some embodiments, the method further comprises, after extracting one or more AP strings from the software application, accessing the AI relevant database and the non-AI relevant database to determine whether any one of the one or more AP strings are stored on either database.

[0026] In some embodiments, the method further comprises, upon determining that any one of the one or more AP strings are stored on either database, retrieving the functionality information from the respective database and excluding the respective AP from the prompt. The report comprises the functionality information retrieved from the respective database.

[0027] In some embodiments, the one or more APs are one or more of:

an API;

an asset file; and

a library.

[0028] In some embodiments, the method further comprises evaluating the machine learning model on a context prompt to provide context to the machine learning model to provide a desired output.

[0029] In some embodiments, the method further comprises, upon classifying one of the one of more APs as AI relevant, generating and transmitting an alert to indicate detection of AI implementation in the software application to a user.

[0030] In some embodiments, the software application is an Android application.

[0031] According to the present disclosure, there is provided software that, when executed by a computer, causes the computer to perform any one of the previously described methods.

[0032] According to the present disclosure, there is provided a system for detecting artificial intelligence (AI) implementation in a software application comprising one or more application packages (APs). The system comprises:

one or more processors configured to:

extract one or more application package (AP) strings from the software application, each of the one or more AP strings representing one of the one or more APs;

create a prompt for a machine learning model, trained to generate output text, comprising the one or more AP strings, the prompt representing instructions to provide a classification of each of the one or more APs and provide functionality information of each of the one or more APs, the classification being AI relevant or non-AI relevant and the functionality information describing a functionality of the respective AP;

evaluate the machine learning model on the prompt to generate output text corresponding to the classification and the functionality information of each of the one or more APs; and

generate a report of the AI implementation in the software application based on the output text.

[0033] Optional features provided in relation to the method, equally apply as optional features to the software and the system.

Brief Description of Drawings

[0034] An example will be described with reference to the following drawings:

[0035] Fig. 1 illustrates a system for detecting artificial intelligence (AI) implementation in a software application.

[0036] Fig. 2 illustrates a method for detecting artificial intelligence (AI) implementation in a software application.

[0037] Fig. 3 illustrates the overall framework and workflow of the preferred embodiment of the disclosed method.

[0038] Fig. 4 illustrates the prompt design framework for the preferred embodiment of the disclosed method.

[0039] Fig. 5 illustrates the distribution of collected apps over the years.

[0040] Fig. 6a shows the total number of packages, APIs, and HTTPS requests after Step 1 (Candidate Extraction) of the preferred embodiment.

[0041] Fig. 6b shows the final determined AI-related packages, APIs, and HTTPS requests after Steps 3-4 (AI capability analysis and detection) of the preferred embodiment.

[0042] Fig. 7 shows a Venn diagram of detected AI apps in the Rule-based approach and the disclosed method.

[0043] Fig. 8 shows the detailed classified tasks and domains of the determined AI candidates.

Description of Embodiments

[0044] This disclosure provides an approach to detecting AI implementation in software applications, which addresses the limitations of current methods, such as rule-based approaches. The disclosed system and method combine program analysis techniques with the power of a machine learning model trained to generate output text, such as, but not exclusively, advanced Large Language Models (LLMs), like ChatGPT (https://chat.openai.com/) to provide a

'lightweight' analysis of AI implementation ('lightweight' referring to the minimal computational and manual resources required). The disclosed system and method extract AI candidate information from a software package file, such as an Android Package Kit (APK) file for the Android operating system, and subsequently queries the machine learning model to confirm AI implementation details. The determined AI information is then incorporated to generate a comprehensive summary regarding the AI implementation including where in the application (e.g., where in the source code) the AI is implemented.

[0045] Automating the human-like semantic-based AI Discovery can address some of the aforementioned challenges of current AI detection methods, where in-wild AI relevant AP candidates can be characterized by understanding their names. In particular, machine learning models, such as currently available LLMs, can act and reason like human experts, which is utilised by the disclosed system and method to aid the AI Discovery process. Furthermore, machine learning models can automate various software tasks, including automatic program repair and code generation and help analyse software efficiently. As such, machine learning models essentially have a substantial amount of knowledge in software practice. As a result, machine learning models, such as LLMs, can be exploited for AI Discovery inside software applications, due to their vast knowledge and understanding of software applications.

[0046] Experiments on sampled Android apps show that the disclosed method can achieve a higher performance in AI apps identification compared to the traditional rule-based approach and also the majority AI implementations in Android apps concern the domains of computer vision and machine learning and data analysis. Moreover, a human study was conducted with six participants regarding the differences between the app developers' provided summary and the tool generated one. The study revealed that the summary generated by the disclosed method is much more acknowledged by these participants. As such, the disclosed method provides a useful and informative summary of the AI implementation, which is far superior to what a human developer can produce. This innovative approach addresses the shortcomings of existing tools, providing a more versatile and insightful method for understanding and guiding the development of AI-powered apps.

[0047] While this disclosure, including the methodology and results, is directed towards a mobile applications and more specifically Android applications, it is noted that the system and method described herein are not limited to mobile applications, such as Android applications. The disclosed system and method are applicable to other software applications deployable on a

variety of different devices running operating systems such as, but not limited to, Apple's iOS, MacOS, watchOS and iPadOS, Microsoft Windows, ChromeOS and Linux.

System

[0048] Fig. 1 illustrates an example system 100 for detecting artificial intelligence (AI) implementation in a software application. The software application comprises one or more application packages (APs). The one or more APs may be one or more of an application programming interface (API); an asset file; and a library. Fig. 1 is one example of a configuration of system 100. However, system 100 is not strictly limited to this configuration and this may be one possible embodiment of system 100.

[0049] System 100 comprises a device 101, which may be smartphone, computer, tablet, a server device, or any other similar device. Device 101 comprises processor 102 connected to program memory 103 and data memory 104. Program memory 103 is a non-transitory computer readable medium, such as a hard drive, a solid-state disk, or CD-ROM. Data memory 104 may be volatile memory, such as cache or RAM, or non-volatile memory, such as an optical disk drive, hard disk drive, storage server or cloud storage, or a combination of both volatile and non-volatile memory.

[0050] Software, that is, an executable program stored on program memory 103 causes processor 102 to perform methods for detecting AI implementation in a software application. When the singular of "processor" is used herein, it is meant to also encompass multiple processors that are individually or together configured (e.g., programmed) to perform the methods disclosed herein. As such, processor 102 may refers to multiple computer processing units (CPUs) and/or graphical processing units (GPUs) that are configured to collectively perform the methods disclosed herein.

[0051] Once executed, the software may cause processor 102 to extract one or more application package (AP) strings from the software application, create a prompt for a machine learning model, trained to generate output text, comprising the one or more AP strings, evaluate the machine learning model on the prompt, and generate a report of the AI implementation in the software application.

[0052] Data memory 104 may store software applications, APs, APs strings, prompts for a machine learning model and the output of the machine learning model, such as output text. The software applications may be stored on data memory 104 in the form of an EXE, APK, BAT, COM, CMD, INF, IPA, OSX, PIF, RUN, WSH or another equivalent data file type. The APs may be stored on data memory 104 in the form of a JSON, XML, and YAM or another equivalent data file type. The AP strings, prompts for a machine learning model and the output text of the machine learning model and the other quantities used to perform the disclosed methods may be stored on data memory 104 in the form of a TXT file, XML file, JSON file or another equivalent data file.

[0053] It is to be understood that any receiving step may be preceded by processor 102 determining or computing the data that is later received. For example, processor 102 may create a prompt for a machine learning model comprising the one or more AP strings, then store the prompt on data memory 104, such as on RAM or a processor register. Processor 102 then requests the data (i.e., the prompt) from the data memory 104, such as by providing a read signal together with a memory address. The data memory 104 provides the data as a voltage signal on a physical bit line and processor 102 receives the prompt via a memory interface.

[0054] System 100 further comprises databases 105, 106 which may store output text generated by the machine learning model and APs strings. Databases 105, 106 may also store other quantities used to perform the disclosed methods. As such, databases 105, 106 may have similar functionality to data memory 104 but located external to device 101. In other embodiments, databases 105, 106 may be located internally to device 101. Processor 102 may establish a communication with databases 105, 106 using I/O port 107. In one example, processor 102 sends a request for the functionality information of one or more APs to databases 105, 106, databases 105, 106 processes the request and send the functionality information to processor 102. Processor 102 may communicate with databases 105, 106 using Structured Query Language (SQL).

[0055] In some embodiments, database 105 stores data regarding AI relevant APs. For example, database 105 may store each AP string of APs classified as AI relevant and respective functionality information. As such, database 105 may be considered as a “AI relevant database”. Similarly, in some embodiments, database 106 stores data regarding AI relevant APs. For example, database 106 may store each AP string of APs classified as non-AI relevant and respective functionality information. As such, database 106 may be considered as a “non-AI

relevant database". While databases 105, 106 are illustrates as two distinct entities in Fig. 1, it is noted that databases 105, 106 may also be a single database or a partition of a single database, for example.

[0056] Processor 102 may establish a communication with databases 105, 106 using I/O port 107 through a wireless connection, such as by using a Wi-Fi network according to IEEE 802.11. The Wi-Fi network may be a decentralised ad-hoc network, such that no dedicated management infrastructure, such as a router, is required or a centralised network with a router or access point managing the network. Processor 102 may establish a communication with databases 105, 106 using I/O port 107 through a wired connection, such as Ethernet.

[0057] Although I/O port 107 is shown as single entity, it is to be understood that any kind of data port may be used to receive data, such as a network connection, a memory interface, a pin of the chip package of processor 102, or logical ports, such as IP sockets or parameters of functions stored on program memory 103 and executed by processor 102. The parameters of functions may be stored on data memory 104 and may be handled by-value or by-reference, that is, as a pointer, in the source code.

[0058] System 100 further comprises server 108. Server 108 may run an application in which a user can input a software application and causes server 108 to perform the method disclosed herein to detect AI implementation in the software application. Server 108 may run the backend of this application, where the frontend corresponds to a graphical user interface which is displayed on device 101, for example. Server 108 may also run the machine learning model. As such, a user may upload a software application to device 101, processor 102 may then extract one or more application package (AP) strings from the software application and create a prompt for a machine learning model comprising the one or more AP strings. Processor 102 may then communicate the prompt to server 108, where server 108 evaluates the machine learning model on the prompt to generate output text. Server 108 may then communicate the output text to processor 102.

[0059] The software applications may be received from a source external to system 100, such as another system located remotely to system 100, which may be in communication with the server 108. The remote system may be located within the same facility as system 100 or may be completely remote from system 100. It is noted that server 108 may have similar functionality to that of device 101. Server 108 may perform parts of the disclosed methods described herein. For

example, server 108 may comprise program memory which comprises software, that is, an executable program stored on program memory, and causes server 108 to evaluate the machine learning model on the prompt to generate output text, as described above. Processor 102 may establish a communication with server 108 using I/O port 107. Server 108 may also be in direct communication with databases 105, 106.

[0060] Software may provide a user interface (such as a graphical user interface) presented to the user on device 101. The user interface is configured to accept input (via buttons or text fields etc.) from the user, via a touch screen or a device attached to device 101 such as a keyboard or computer mouse. These devices may also include a touchpad, an externally connected touchscreen, a joystick, a button, and a dial. In an example, device 101 may display several software applications. The user may select a software application to detect AI implementation by interacting the touch screen or keyboard, which causes processor 102 to perform the method disclosed herein on the selected software application. The user interface may also display the output text generated by the machine learning model and/or the report of the AI implementation in the software application based on the output text.

[0061] It is noted that system 100 of Fig. 1 is only meant to illustrate an example and a preferred system which is capable of performing the disclosed method. Many other configurations of system 100 may equally perform the disclosed method.

Method

[0062] Fig. 2 illustrates method 200 for detecting AI implementation in a software application, where the software application comprises one or more application packages (APs). Fig. 2 is to be understood as a blueprint for a software program and may be implemented step-by-step, such that each step in Fig. 2 is represented by a function in a programming language, such as, but not limited to, Python, C++ or Java. The resulting source code is then compiled and stored as computer-executable instructions on program memory 103, which causes processor 102 (or multiple processors or a distributed computing architecture such as the server 108) to perform method 200.

[0063] To begin method 200, processor 102 extracts 201 one or more AP strings from the software application, where each of the one or more AP strings represent one of the one or more APs. An AP string may be sequence of characters that identifies the corresponding AP, such as

the title or name of the AP, a Hypertext transfer protocol secure (HTTPS) request, a file name of the corresponding AP or another equivalent text-based identifier. For example, the software application may comprise the Object Detection and Tracking package (https://firebase.google.com/docs/ml-kit/object-detection) and may extract the AP string 'com.google.firebase.ml.vision.objects' from the software application by analysing the software application's source code. If the AP is a library or a library file, the corresponding AP string may be the library name or some strings extracted from the library, such as a text-based identifier to distinguish the specific library.

[0064] Processor 102 may determine that an AP string corresponds to an AP by determining that the software application calls the AP in the source code. For example, processor 102 may determine an AP string corresponding to an AP by identifying a HTTPS request in the source code. In some examples, the AP string may be equivalent to the HTTPS request (i.e., for the Object Detection and Tracking package, the AP string may be "https://firebase.google.com/docs/ml-kit/object-detection").

[0065] Processor 102 may also extract 201 one or more AP strings from the software application by applying an analysis method, which is able to detect the one or more AP strings. For example, processor 102 may apply a static analysis framework method, which can extract 201 the APs strings directly from the application file, such as an APK file in the case of an Android application.

[0066] Processor 102 then creates 202 a prompt for a machine learning model, trained to generate output text, comprising the one or more AP strings. The prompt represents instructions to provide a classification of each of the one or more APs and provide functionality information of each of the one or more APs, where the classification is either AI relevant or non-AI relevant and the functionality information describes a functionality of the respective AP. In some examples, the prompt (or any other prompt described herein) may be, at least partially, generated using a machine learning model, which may be similar to the machine learning model in which the prompt is evaluated. For example, a LLM such as ChatGPT may, at least partially, generate a prompt which is then evaluated on the same LLM.

[0067] The machine learning model is understood to be a model, such as a mathematical model, that receives input text (as referred to as 'prompts') or other inputs such as, but not limited to, image data and audio data and generates an output based on the input. The machine

learning model may be of an architecture, such as, but not limited to, a neural network, for example. In general, machine learning models are 'trained' to learn and recognise patterns in an input and provide an output that is a prediction based on the training it has undergone. Training involves updating weights or parameters (as referred to as hyperparameters) of the machine learning model, which define the machine learning model, to minimise a loss value, thereby creating a trained machine learning model (in other words, a machine learning model trained to generate an output). This may involve a gradient descent and backpropagation method.

[0068] The machine learning model may be stored on data memory 104 or server 108 by storing the weights that define the model. As such, the machine learning model may be referred to as a "memory model", given that it is defined by parameters (i.e., the weights) which can be stored on computer memory. In some embodiments, the machine learning model may be programmed on an integrated circuit, such as a field-programmable gate array (FPGA) or an NVIDIA processing unit. In such an embodiment, processor 102 may not retrieve the parameters from data memory 104. Instead, an input may be communicated from processor 102 to the integrated circuit and the integrated circuit may apply the machine learning model to the input and generate an output, which is then communicated to processor 102.

[0069] Integrated circuits, such as FPGAs, can be used where flexibility, speed, and parallel processing capabilities are desired. In such an embodiment, the integrated circuit may be part of device 101 of system 100 and may be considered as a "processor" or "processing unit", similar to processor 102. Other implementations, such as application specific integrated circuits (ASIC) or neuromorphic architectures are equally useable.

[0070] There may be a number of different ways to invoke the machine learning model, which all fall within the meaning of "evaluating the machine learning model" as recited throughout this disclosure. For example, evaluating the model may involve calling an API routine to send the prompt to a server and the server then performs the calculations according to the trained machine learning model and returns the results. In other examples, evaluating may involve issuing a command to local hardware, such as a local chip, device, machine learning accelerator (e.g., a USB device design to efficiently perform machine learning tasks or NVIDIA's Deep Learning Accelerator (DLA)), etc., that has the trained machine learning model stored thereon and provides a command interface to interact with the model. It is also possible to have a local copy of the machine learning model available so that the calculations are performed by the main

processor of the local machine. Other local, remote or distributed implementations (such as cloud computing environments) are equally useable.

[0071] In some embodiments, the machine learning model may be trained on a broad range of different data such that it can be applied across a wide range of use cases. Such a machine learning model may be referred to as a "foundational model". Some foundational models that are applicable to the disclosed method include those that are publicly available and/or trained on publicly available data. In other embodiments, the machine learning model may be trained on a specific set of training data, in order to focus the generated outputs of the machine learning model to a specific task or area of interest. In yet other examples, the foundational model is further trained on the specific set of training data to improve the model in the area of interest.

[0072] In some embodiments, the machine learning model is a multimodal machine learning model, in which multiple inputs of different modalities (e.g., text, image data and audio data) are used to provide one or more generated outputs. An example of a multimodal machine learning model is an object detection model, which detects the location of a specific object (specified by input text, for example) in an image. This example model may generate output text that describes the location of the specified object in the image. Although the multimodal machine learning model can be evaluated on multiple input of different modalities, the multimodal machine learning model can also be evaluated on a single input and still generate an output based on the single input.

[0073] The machine learning model may be trained to generate output text based on input text and hence, may be a chat-based machine learning model. The machine learning model may also be referred to as a trained generative language model. Both the input and output text may be in the form of "natural language" (i.e., any language that occurs naturally in a human community by a process of use, such as spoken English, for example). Such machine learning models may be referred to as "chatbots". A chatbot (which may also be referred to as a chatterbot) is designed to mimic human conversation (using natural language) through text or voice interactions. More particularly, the chatbot responds to input natural language using output natural language. Examples of such chatbots currently include ChatGPT (using GPT-3 or GPT-4), Microsoft's Bing Chat (which uses OpenAI's GPT-4) and Google's Bard.

[0074] In more specific examples, the machine learning model may be an LLM. A LLM is a type of artificial intelligence system, characterised by its massive training data and high volume

hyperparameters. These language models ingested input text sourced from various sources and use fine-tuning to predict potential tokens or words. This enables them to perform various natural language processing tasks, including sentiment analysis, document classification, and lexical analysis, among others. However, their capabilities have extended beyond these tasks to encompass a broader range of applications and industries, such as chatbots, content generation, and even scientific research, demonstrating their versatility and growing significance in the field of AI.

[0075] LLMs like GPT (i.e., GPT-3, GPT-3.5 and GPT-4) and LLaMA are built upon the transformer architecture. These models are pre-trained on vast volumes of text data collected from publicly available online resources, and have been specifically trained to understand and process natural language. However, they differ from traditional natural language processing (NLP) systems in how they handle NLP processing tasks. Instead of relying solely on pre-training and fine-tuning, LLMs excel in NLP tasks through the technique of prompt engineering. In prompt engineering, tasks are conveyed as text descriptions, and these descriptions are presented to the model for interpretation and the generation of corresponding responses.

[0076] The LLM may be an artificial neural network, such as transformer model (e.g., a generative pre-trained transformer) which utilises encoder and decoder networks. These LLM are (pre-) trained using self-supervised learning and semi-supervised learning. In essence, LLMs are trained to predict what word comes next in a sequence of words which can be based on the semantic closeness of the words. LLMs form these predictions by 'tokenising' the words in the output text (i.e., converting the words into a vector of numbers). These tokens can also incorporate other information such as the position of the word in the input text and the information about the adjust words.

[0077] The process of creating these tokens is also referred to as 'encoding' or 'semantic encoding', as information regarding each word is essential encoded into a vector of numbers. The process of creating these token that incorporate the position of each word in the input text is referred to as "positioning embedding". The encoding process also includes embedding at least one learnable parameter, which is determined through training. The opposite process then occurs where the tokens are converted from a vector of numbers into words, which is referred to as 'decoding'. This gives the final output of the LLM in the form of text.

[0078] The machine learning model may also comprise an attention mechanism that applies weights to the tokens and may be characterized by its self-attention layers. These layers enable the model to assess the significance of words in an input relative to each other, thereby providing a more nuanced comprehension of the text. The attention mechanism may also include a scaled dot-product between different matrices generated by the model to calculate the weights. In simple terms, the attention mechanism allows a model to focus on different parts of the input when generating each element of the output. This dynamic focusing capability results in a more contextually aware model, producing better results in tasks like translation, summarization, or text generation. Other ways of achieving attention within the model would be equally possible.

[0079] As a result, the machine learning model can process a large number of input values, such as an input text paragraph, at one time rather than sequentially in order to consider the context of each word. Nevertheless, the overall number of parameters in the machine learning model is relatively large, which is the reason those models are referred to as large models, such as LLMs or Large Action Models (LAMs). In some examples, a model is large if it has more than 100 million parameters or more than 1 billion parameters or more than 1 trillion parameters.

[0080] In some embodiments, the machine learning model referred to in this disclosure may be multiple machine learning models that have a 'global' input and a 'global' output. In that sense, the multiple machine learning models may be "daisy chained" together, such that the output of one machine learning model becomes the input for the next machine learning model within the chain. Some of the multiple machine learning models may operate in parallel, rather than in a series or chain. Each of the multiple machine learning models may have its own memory and/or have access to a common memory that is shared become some or all of the multiple machine learning models. In this sense, the multiple machine learning models may resemble parallel processing or parallel computer, such a computer architecture with multiple CPU cores which can be operated in parallel.

[0081] The prompt of step 202 may be instructions in the form of a text that is created by processor 102. For example, after extracting the string related to the Object Detection and Tracking package, processor 102 may create a prompt indicative of the following text: "analyse the functionality of the application package given by the string 'com.google.firebase.ml.vision.objects' and detect the application package's relevance to artificial intelligence". The text "analyse the functionality of a list of the application package" represents instructions to provide functionality information of the AP, while the text "detect the

application package's relevance to artificial intelligence" represents instructions to provide a classification of the AP. Processor 102 may create 202 a prompt that is specifically tailored towards the machine learning model used. As such, the prompt may vary based on the machine learning model.

[0082] Processor 102 then evaluates 203 the machine learning model on the prompt to generate output text corresponding to the classification and the functionality information of each of the one or more APs. In some embodiments and as described earlier, server 108 may evaluate 203 the machine learning model on the prompt, rather than processor 102. For example, server 108 may store the machine learning model and receive the prompt from processor 102. Server 108 may then evaluate 203 the machine learning model on the prompt to generate output text and communicate the output text to processor 102.

[0083] Finally, processor 102 generates 204 a report of the AI implementation in the software application based on the output text. In some examples, the report may simply be the output text from the machine learning model. In other examples, the report may provide other information based on the output text, such as the classification of each of the one or more detected APs in the software application as AI relevant or non-AI relevant, the functionality of each of the one or more detected APs and where the APs are being implemented in the software application (e.g., where in the source code each AP is being called). In particular, processor 102 may generate 204 the report in such a way that is easily interpretable by a user or visually appealing.

[0084] In some embodiments, the prompt of method 200 as well as any other prompts described herein comprise learning examples indicative of a desired output from evaluating the machine learning model on the prompt. This enables the machine learning model to understand the format of the desired output to produce an output that is similar to the learning examples. The learning examples may be zero-shot learning examples. Zero-shot learning represents a learning paradigm that leverages the capabilities offered by LLMs. In this paradigm, task requirements are directly presented to LLMs, and they, in turn, provide answers, i.e., predicted tokens or responses, as outputs.

[0085] To increase the validity of the predicted answer regarding the aforementioned complex and unobserved tasks, in some examples, the learning examples may be few-shot learning examples. The few-shot learning approach completes the learning tasks by first learning from a few examples (i.e., the supervised information) showing the sample task descriptions and desired

output answers and then predicting the output with the given real task description. With only a limited number of examples, it could quickly adapt to new tasks and/or concepts, which makes few-shot learning highly applicable in various domains, including computer vision, natural language processing, and even robotics, where the ability to generalize from minimal data is crucial.

[0086] In some embodiments, processor 102 or server 108 evaluates the machine learning model on a context prompt to provide context to the machine learning model to provide a desired output. For example, the context prompt may be indicative of the following text: "You are a helpful assistant to answer queries related to utilising APIs, libraries, and some specific files". The context prompt thereby directs the machine learning model to provide an output that is desired for the purpose of detecting AI implementation.

[0087] In some embodiments, upon classifying one of the one of more APs as AI relevant, processor 102 generates and/or transmits an alert to indicate detection of AI implementation in the software application to a user. In other words, processor 102 transmits a message or alert to a user device to alert (or bring to the attention of) the user of the detection of AI implementation in the software application. In other embodiments, processor 102 generates a graphical user interface and, upon classifying one of the one of more APs as AI relevant, processor 102 generates an alert on the graphical user interface to indicate detection of AI implementation in the software application to a user of the graphical user interface. Processor 102 may also block the use of the APs detected as AI relevant, in which user of device 101 cannot use these APs when developing software. Processor 102 may also provide a report of possible vulnerabilities based on the APs detected as AI relevant.

[0088] It is noted that the machine learning model may be evaluated on multiple prompts throughout the disclosed method and especially in the preferred embodiment, as will be described below. In each evaluation instance, the machine learning model may have 'memory' or 'context' from a previous prompt evaluation and, as such, the generated output may be based on this 'memory' or 'context', as described above with the context prompt. However, in some embodiments, one or more evaluation instances may not use 'memory' or 'context' from a previous prompt evaluation.

Preferred embodiment

[0089] Fig. 3 illustrates the overall framework and workflow of the preferred embodiment of the disclosed method. It can be seen that Fig. 3 comprises databases 105, 106 from Fig. 1. The preferred embodiment comprises five tasks: AI app filter, candidate extraction, AI candidate determination, AI capability analysis, and AI service summarization. These tasks are achieved by integrating static analysis with the powerful AI advance, ChatGPT. ChatGPT is used in the preferred embodiment as it is current the best, publicly available LLM. However, it is noted that the disclosed method, including the preferred embodiment, can be used with other LLMs, especially those yet to be developed. In particular, the disclosed method, can applied to any model that has in-context learning capability. In the preferred embodiment and with reference to Fig. 4, there are three offline steps (Steps 1-2 and Step 5) and four online steps (Step 0, Steps 3-4 and Step 6).

[0090] While the preferred embodiment of the disclosed method will now be described, it is understood that other embodiments of the disclosed method, which use any combination of the features described hereafter, are equally possible. The preferred embodiment is also directed towards detecting AI implementation in Android applications. However, it is noted that other embodiments may be directed towards other types of software applications. The preferred embodiment is also directed using ChatGPT as the machine learning model. However, it is noted that other embodiments may be directed towards other machine learning models.

AI App Filter

[0091] To locate the potential AI apps, previous methods adopt manual inspection of the app candidates (i.e., the software application that potentially implement AI). However, this process is time-consuming. To fasten the possible AI implementation in a software application, it is possible to determine whether a software application uses AI techniques or not without detailed inspection. Conventionally, software applications are usually released with a brief description of its functionalities for users. For example, the description of Wise AI Chatbot (https://play.google.com/store/apps/details?id=com.jgapps.chatgpt) mentions that 'Wise Chat GPT Pro is the first app to utilize the advanced technology of ChatGPT 3.5 Turbo to provide instant and [intelligent] responses. Boost your productivity with Wise Chat GPT Pro, our [AI] [chatbot] powered by ChatGPT 3.5 Turbo technology, which delivers interactive and fun conversations that will make you more productive.'. Such AI-related applications usually use AI-related terms in the description.

[0092] As such, the description of a software application can provide information regarding whether the application uses AI techniques. This can be utilised to filter applications that are unlikely to use AI implementation, rather than applying method 200 on a software application that is already known to not implement AI. This provides more computational efficiency to the disclosed method as only the potential applications that use AI techniques are further analysed by applying method 200.

[0093] As such, in the preferred embodiment, processor 102 analyses a description of the software application before extracting one or more AP strings from the software application to identify the one or more APs. In the preferred embodiment, the process of analysing an application's description to determine AI implementation is conducted into a sequential two-staged process: (1) rule-based filtering firstly and then (2) LLM-based approach to identify the apps leveraging AI technologies.

[0094] Firstly, processor 102 analyses the description of the software application by performing a keyword search using terms relevant to AI. Secondly, the machine learning model is utilised provide further information from the description of the software application. As such, processor 102 creates an initial prompt based on the description of the software application, where the initial prompt represents instructions to provide a classification of the software application, the classification of the software application being likely to implement AI or unlikely to implement AI. The process of analysing an application's description to determine AI implementation is referred to as "AI App Filter".

[0095] In more detail, the filter employs keyword extraction rules to swiftly and accurately identify explicit AI-related terminology within application descriptions in the first stage. A wide range of terms are defined from high-level functionalities (e.g., "AI" and "Deep Learning") to low-level ones (e.g., "text classification" and "facial recognition"). By capturing exact matches, the filter effectively narrows down the pool of applications for more intensive analysis, ensuring that subsequent stages focus on high-potential candidates. This step ensures that only descriptions with genuine relevance to the domain are shortlisted, filtering out the noise and irrelevant entries.

[0096] However, it is possible that some descriptions contain these predefined terms but do not use them to describe their main functionalities. To better understand the app descriptions, the machine learning model (i.e., ChatGPT in the preferred embodiment) is used to classify if the

software application will likely use AI techniques based on the given description. The corresponding prompt is indicative of the following text: "Analyse the app's description and determine if potential AI services are provided by the app.". The integration of ChatGPT addresses the limitations of rule-based methods by adding a layer of semantic depth, which is essential for the comprehensive identification of AI-integrated applications.

[0097] As such, in the second stage of the AI App Filter, once the preliminary filtration is achieved, the shortlisted descriptions are subjected to a more nuanced analysis using ChatGPT via the OpenAI API. This LLM model delves deeper into the context, semantics, and underlying intent of the descriptions. Its capacity to understand subtleties in language ensures that apps are not merely retained based on keyword presence but are genuinely pertinent to AI and deep learning in their functionality and design.

Candidate Extraction

[0098] After locating the potential apps that use AI techniques, all necessary information (or candidates) from the source file, such as an Android APK file, of a given software application. In the preferred embodiment, processor 102 extracts 201 the one or more AP strings from the software application by applying a static analysis method to the software application. Specifically, in the preferred embodiment, the static analysis technique "App Scanner" (https://ieeexplore.ieee.org/document/9402104) is used to extract APIs from an APK file and implemented based on the popular static analysis framework, Soot (https://soot-oss.github.io/soot/). The traditional Java-based static analysis framework, Soot, was enhanced with the ability to convert Dalvik bytecode (https://source.android.com/docs/core/runtime/dalvik-bytecode) into Jimple's three-address code. Dalvik bytecode is the default executable representation for Android apps. It is indirectly compiled from Java code due to the running environment with less memory and power provision, while Jimple is the default internal representation in Soot.

[0099] By converting Dalvik bytecode into Jimple representation, App Scanner is improved to additionally identify HTTPS requests defined in the source code, and extract asset files and native libraries, which are always leveraged to implement on-device AI inferences, such as the extracted library libtensorflow_inference.so, the package com.google.firebase.ml.vision.objects, and the HTTPS requests https://api.openai.com/v1/chat/completions. Concerning the extracted native libraries, HTTPS requests are also drawn out from these native ones using readelf

(https://man7.org/linux/man-pages/man1/readelf.1.html) and objdump (https://man7.org/linux/man-pages/man1/objdump.1.html) as some applications implement AI models in native code, such as C/C++, and invoke such inferences via NDK (https://developer.android.com/ndk).

Knowledge Base Interaction

[0100] As there are many overlapped candidates used in various apps, repeatedly analysing them can be very costly for a computational perspective. Therefore, a knowledge base (KB) is constructed to store all candidates that have already been analysed in the previous apps, so as to speed up the determination of extracted candidates. Specifically, two KBs are built: (1) AI KB and (2) non-AI KB, to store all candidates and their analysis results. While AI KB aims to help provide all capability analysis for AI service summarization, non-AI KB mainly helps reduce the number of irrelevant candidates before analysis. Once the candidates are extracted from the given APK file, they will be examined in both KBs. The analysis and detection results will be returned if they have been analysed. Otherwise, they will be passed to the next step (i.e., Step 3 and Step 4 of Fig. 4), and after the step of AI candidate detection, these analysed candidates will be populated to the corresponding KB (i.e., Step 5 of Fig. 4).

[0101] More specifically, processor 102 stores each AP string of APs classified as AI relevant and respective functionality information on an AI relevant database and storing each AP string of APs classified as non-AI relevant and respective functionality information on a non-AI relevant database. With reference to Fig. 1 and Fig. 3, database 105 represents the AI relevant database, while database 106 represents the non-AI relevant database. As a result, the preferred embodiment essentially builds a library of AI-relevant and non-AI relevant APs. It is noted that previous methods of detecting AI implementation struggle to build and maintain such libraries, due to the time-consuming nature of the work and new AI relevant APs being continuously released.

[0102] In the preferred embodiment, after extracting one or more AP strings from the software application, processor 102 accesses the AI relevant database and the non-AI relevant database to determine whether any one of the one or more AP strings are stored on either database. Then, upon determining that any one of the one or more AP strings are stored on either database, processor 102 retrieves the functionality information from the respective database and excluding

the respective AP from the prompt. The generated report then contains the functionality information retrieved from the respective database.

[0103] By retrieving the functionality information from the respective database and excluding the respective AP from the prompt, processor 102 does not need to apply the remainder of method 200 to APs that have already been analysed and catalogue in the databases. This provides more computational efficiency to the disclosed method. The disclosed method addresses a long-standing limitation of current methods as it is expensive (from a computer perspective and time perspective) to collect AP information, build and maintain a database or catalogue of APs, which includes not just common APs but also incorporates the evolutions of new APs. Current attempts to create such a database or catalogue found difficultly in store and manage such information, due to computational resources and manual inspection required.

AI Capability Analysis and Detection

[0104] The emphasis on proper structural prompt design is important when aiming for higher precision in AI determination. In particular, it can be difficult to understand how to effectively prompt some machine learning models as these models are often offered as "black boxes". In machine learning, a black box refers to giving a prediction (such as output text) based on an input without explaining or showing how the machine learning model has reached its precision. This is common with publicly available machine learning models. Therefore, such effectively design is desirable as it ensures that the machine learning model has a clear understanding of the task at hand, thus optimizing its response.

[0105] However, even with an engineered prompt, it could still give out an inaccurate answer with respect to complicated task descriptions. In particular, it has been observed that extensive and intricate directive prompts often lead to inaccurate prediction results. This is likely because this would be an "epic prompt" which may overwhelm the machine learning model. Such an epic prompt also has other disadvantages such as being. hard to debug, and only sometimes provides the desired output.

[0106] To provide a more accurate answer to the sophisticated tasks, in the preferred embodiment, processor 102 creates 202 the prompt for the machine learning model by creating a first prompt and a second prompt. This is based on a "divide-and-conquer" approach, in which the problem is decomposed into several intermediate steps and the final answer will be given out once these intermediate steps are solved successfully. This may also be referred to as 'chaining'.

[0107] Decomposing the problem into several intermediate steps provides much more accurate results as the machine learning model can focus on a specific, intermediate task, rather than focusing on the broad, overall task. This approach also resembles a structural prompt design strategy, drawing inspiration from the "Single Responsibility Principle" (SRP) of Object-Oriented Design. This approach ensures that each prompt is tailored to execute a singular and specific task, thereby enhancing the clarity and precision of the model's output. As such, each step in the chain is a single responsibility, giving the chain of prompts a modular design.

[0108] In the preferred embodiment, the first prompt represents instructions to provide the functionality information of each of the one or more APs and the second prompt represents instructions to provide the classification of each of the one or more APs. As such, in the preferred embodiment, server 108 evaluates 203 the machine learning model on the prompt by evaluating the machine learning model on the first prompt to generate initial output text and evaluating the machine learning model on the second prompt to generate the output text, where the second prompt is based on the initial output text.

[0109] While in some embodiments, server 108 evaluates 203 the machine learning model on the prompt by evaluating the machine learning model on the second prompt then evaluate the first prompt based on the output of evaluating the second prompt, the result described herein show that there is an advantage to evaluating the first prompt then the second prompt, as the machine learning model (i.e., a LLM such as ChatGPT in the preferred embodiment) can sometimes misclassify some key information without the reasoning that is provided by the functionality information.

[0110] In more detail, to make ChatGPT reliably complete the task of AI candidate determination, the overall task is split into two steps: (1) AI capability analysis (i.e., Step 3 of Fig. 4) and candidate detection (i.e., Step 4 of Fig. 4). After collecting all deduplicated candidates from Step 2 of Fig. 4, ChatGPT is prompt to generate the analysis emphasizing the functionality. For example, ChatGPT will provide the analysis of 'Google's ML Kit API for object detection and tracking in images and videos to the package com.google.mlkit.vision.objects. In addition, there might be some obfuscated or customized candidates that ChatGPT has no knowledge of, but it still could provide meaningful insights. This is especially useful as many new AI relevant APs are developed and released.

[0111] For instance, the package nlp.WordPieceModelPB can be hard to determine with rule-based methods. With the help of ChatGPT, the human-like speculation of 'A library for tokenizing text using the WordPiece algorithm, implemented in Protocol Buffers format' can effectively describe the highly potential functionality. Based on the generated analysis, ChatGPT is further prompted to classify if the candidate uses AI techniques. As ChatGPT can understand its generation well, the steps of AI capability analysis and AI candidate detection are straightforward by adopting the prompts of Step 3 and Step 4 of Fig. 4.

AI service summarisation

[0112] The last step is AI service summarization, where a brief report is generated to describe what AI services the software application uses via the identified AI candidates. As ChatGPT and most LLMs always have limited context length, it is challenging to put all the determined AI candidate names and their corresponding analysis into the same context window. To address this limitation, the API-level candidates are excluded, as their functionalities are likely to be implied by their packages. As such, in the preferred embodiment, processor 102 classifies each of the one or more APs as an application programming interface (API) or a non-API and creates a third prompt further represents instructions to exclude APs classified as an API from the report.

[0113] By concatenating all determined AI candidate analyses except the APIs, ChatGPT is prompted to generate a brief AI service report, which is shown in Step 6 of Fig. 4. As such, processor 102 creates the third prompt based on the output text, the third prompt representing instructions to generate the report; and generating a report comprises evaluating the machine learning model on the third prompt to generate the report.

Experimental Configuration

[0114] The preferred embodiment was used to produce the results described in the following section. The experimental configuration used to produce the results described in the following section will now be described. To make ChatGPT fully understand Steps 3, 4 and 6 of Fig. 4, five input-output examples are provided to the LLM for the few-shot in-context learning. For the actual implementation of the disclosed method, the default 4k context windows are used. As ChatGPT has limits for requests per minute, it is time-consuming to process one app with one candidate per request. However, when inputting too many candidates in one request, ChatGPT can likely miss a few candidates in the outputs and cause misalignment.

[0115] To balance task accuracy and processing speed, a batch size of 3 candidates is used and instructed ChatGPT to output in a JSON format, which can be easily parsed. The parameters of the model are mainly kept at their default values, except for the temperature and TopP parameters, which were adjusted from the default value of 1 to 0.2 and 0.95 to reduce the impact of ChatGPT's output randomness but maintain the creativity to generate candidate analysis and app summary.

Evaluation

Aims

[0116] The preferred embodiment was evaluated to address the following questions in light of the aims of this evaluation:

1. *How reliable is the disclosed method in determining AI capability for existing AI apps?* To answer this question, the effectiveness of the disclosed method is examined to determine the AI capabilities of AI Android apps. Meanwhile, the possibility of making the disclosed method more efficient is explored and trade-offs between processing time and reliability are presented. By answering this question, the efficacy of the disclosed method can be provided.
2. *How reliable is the disclosed method in determining AI capability for apps in the wild?* To answer this question, the disclosed method is compared to a rule-based approach on Android apps based on their performance determining AI capability of in-wild Android apps. By answering this question, the strengths of the disclosed method in AI app detection are demonstrated.
3. *How useful is the AI summary generated by the disclosed method?* In addition to pinpointing AI candidates and further apps, this is taken one step further by summing up and predict the potential AI capabilities concerning the identified AI candidates in this question. By answering this question, a better understanding of what type of AI is used is developed, which provides a more meaningful and insightful summary report for future app developers and researchers.
4. *What insights can be gained from the disclosed method analysis?* Given the successfully identified apps, the AI tasks and domains among the AI candidates and the determined AI apps are summarised. By answering this question, a further understanding of the detailed AI function provisions in these AI apps can be provided.

Dataset

[0117] The following described experiments were conducted on two separate datasets. The first dataset was collected from AndroZoo (https://androzoo.uni.lu/) and contained 56,682 Android apps. The second dataset was curated by the inventors, and contained 15,517 Android apps, as the first dataset did not contain the newly released apps, especially the ones implemented on newly released LLMs, such as ChatGPT.

[0118] Dataset 1

[0119] The disclosed method was compared to a rule-based approach, which identifies AI-driven Android apps via keywords matching on the dissected and decompiled Android apps. The rule-based approach successfully identified 56,682 AI apps from a total of 7,259,232 apps on AndroZoo (the latest Android version is considered while multiple versions are stored). 56,682 Android apps was used as a baseline to cross-validate the disclosed method. Fig. 5 illustrates the distribution of collected apps over the years.

[0120] Dataset 2

[0121] For the second dataset, AndroZoo was again utilised, where the latest release for every Android app was extracted from its provided CSV file. With these latest application releases, the npm package google-play-scraper (https://www.npmjs.com/package/google-play-scraper) was used by feeding package names to it to determine if the app is still publicly available on Google Play store and, if so, to retrieve the app metadata, especially the app description and release date.

[0122] To determine if the released app is AI-related or not, the AI App Filter process (Step 0 of Fig. 4) was used. Using this process, script marries the precision of rule-based systems with the adaptability and depth of machine learning models. This dual-layered approach guarantees a curated list of apps that genuinely resonate with AI and deep learning principles and applications.

[0123] Following Step 0 (AI App Filter) of the preferred embodiment, shown in Fig. 4,, 15,517 (only the latest released version is considered) Android apps were successfully identified from a total of 23,321,823 apps on AndroZoo. Going one step further, the release date was also extracted from the metadata provided on the Google Play Store and Fig. 5 shows the distribution of the released Android apps over the last decade. It clearly shows the rise of the released apps is

at the same pace as the prevailing AI technologies. It is also worth mentioning that the total number of apps in Fig. 5 is 15,055 due to the release date missing in the given metadata among 462 Android apps, which only accounts for 2.98% (462/15,517). Therefore, the rising trend of AI apps over the past ten years is still valid.

How reliable is the disclosed method in determining AI capability for existing AI apps?

[0124] To address this question, whether the disclosed method can effectively identify AI capabilities for pre-determined AI apps is examined. To do so, dataset 1 was used. However, as dataset1 only determines AI apps but does not provide ground truths of AI candidates, a subset of 382 AI apps were randomly sampled and used to perform the human validation to assess the disclosed method.

[0125] However, it is time-consuming and impractical to check every app in dataset 1 manually. As such, the online sample Size Calculator (https://www.surveysystem.com/sscalc.htm) was used to compute the number of sample apps (with a confidence level of 95% and margin of error of 5%), which provided the 382 apps randomly selected from the total of 56,682.

[0126] Fig. 6 shows the quantity distribution of three representative candidate types, packages, APIs, and HTTPS requests. Fig. 6a shows the total number of packages, APIs, and HTTPS requests after Step 1 (Candidate Extraction) of the preferred embodiment. Fig. 6b shows the final determined AI-related packages, APIs, and HTTPS requests after Steps 3-4 (AI capability analysis and detection) of the preferred embodiment.

[0127] The median values for the total considered packages, APIs, and strings in Fig. 6a are 433.5, 296, and 53.5, respectively. It is noteworthy that the number of packages, APIs, and HTTPS requests are all proactively filtered when extracted from Android APK files via a self-collected whitelist of commonly non-AI candidates. The whitelist of non-AI candidates comprises the official Java and Android kernel-related packages and API signatures.

[0128] The median values for the determined AI packages, APIs, and HTTPS requests are 9.5, 33, and 0, respectively. Compared to the considered candidates, much more packages are removed as they are not reasoned as AI-related candidates while higher number of APIs are kept as AI candidates, which would play an essential role in AI app determination. This can be explained by the fact that most AI APIs only belong to certain AI packages. In terms of HTTPS

requests, they are merely detected by the disclosed method, as the introduction of HTTPS requests for OpenAI is comparatively late and they are not pervasive among the collected experimental apps.

[0129] To examine the reliability of the disclosed method in determining AI candidates on the sampled dataset, multiple authors were asked to independently label 49,371 candidates appearing in the sampled 382 AI apps. Similar to the disclosed method, candidates are classified into two groups: AI candidates and non-AI candidates. True Positive is defined as the candidates that are correctly identified as AI candidates, while True Negative refers to the candidates that are correctly detected as non-AI candidates. After the human validation, a high precision of 98.05% and recall of 93.31% is achieved, with Cohen's unweighted kappa coefficient of 0.82. The results indicate that the disclosed method is highly accurate in AI candidate determination and external observers are very likely to agree on the outcomes of the disclosed method.

[0130] The reliability of the disclosed method in identifying AI apps was then further evaluated. further validate how reliable the disclosed method is in identifying AI apps. An AI app is defined as the one that uses AI candidates. Based on the candidate analysis via the disclosed method, 378 out of the total of 382 apps were identified as AI apps while the remaining 4 as non-AI apps. The reason 4 apps were identified as non-AI ones is that only some file names ending with the keyword .model could be identified, but any complementary AI evidence such as AI-related package names, APIs could not be detected. Without further complementary information, it is difficult to determine what type of AI capabilities and where these abilities are harnessed. Therefore, these apps are not categorised as AI ones.

[0131] During the evaluation of the disclosed method, it was found that the disclosed method is relatively slow as it needs to generate the analysis for both AI and non-AI candidates at first. While the design of Step 3 (AI Capability Analysis) and Step 4 (AI Candidate Detection) is motivated by a "divide and conquer" approach, it may not be optimised for the disclosed method. To efficiently determine the AI capability, one intuitive approach is to classify AI candidates and then provide analysis only for AI candidates. To evaluate the reliability of this approach, the same evaluation settings is followed on the sampled 382 apps. The results of this evaluation is presented in Table 1, which is compared them with the ones from the original implementation.

Analysis/Detection	**Detection/Analysis**

AI Candidate	AI Apps	Time (sec)	AI Candidate	AI Apps	Time (sec)
2859	378	638.76	2192	371	226.94

Table 1: Comparison of the reliability of the disclosed method between the implementations of analysis-then-detection and detection-then-analysis. The time is the average running time for an Android APK in seconds.

[0132] It is first observed that the processing time per Android app is significantly decreased by 65% after adopting the detection-then-analysis implementation. While the new approach is still able to detect most AI apps (371 vs. 378), it is noted that the number of detected AI candidates is much less than the original ones. The observation indicates the trade-off between the reliability and efficiency of the disclosed method. Based on the overall results, it is suggested that detection-then-analysis can only be used when developers are interested in finding AI apps. If developers seek to understand all possible AI capabilities of an app, it is logical that the analysis-then-analysis method is more effective to capture AI candidates.

[0133] In summary, the efficacy of the disclosed method regarding the performance of determining AI capability for existing AI apps is demonstrated. Human validation suggests that the disclosed method can achieve high precision (98.05%) and recall (93.31%) in detecting AI candidates and AI apps.

How reliable is the disclosed method in determining AI capability for apps in the wild?

[0134] Here, the performance of the disclosed method in AI app detection is investigated, as well as its advantages compared to the state-of-the-art rule-based approaches. The performance of a rule-based approach is compared to the disclosed method. To have a fair comparison, the preferred embodiment of the disclosed method is evaluated on dataset 2, which contains 15,517 Android apps released on the Google Play Store. However, it is time-consuming to experiment on all the collected Android apps. To select a representative subset, the focus is first put on Android apps published over the past five years from 2019 to 2023, which will result in a total of 12,740, and then randomly pick 30% apps for each year. In total, there are 3,819 $(1980 * 30\% + 2336 * 30\% + 2326 * 30\% + 3053 * 30\% + 3044 * 30\%)$ Android apps for the experiment.

[0135] Following the same setting as in the previous section, the precision and recall regarding the AI candidate identification are computed. The disclosed method shows the precision of 97.32% and recall of 91.01%. The inter-annotator reliability measured by Cohen's unweighted kappa coefficient is 0.80, indicating the high agreement with outcomes of the disclosed method.

Rule-based approach		**Disclosed method**	
AI Apps	Non-AI Apps	AI Apps	Non-AI Apps
603	3216	1366	2453

Table 2: Number of AI and Non-AI apps classified by the rule-based approach and the disclosed method.

[0136] Table 2 represents the detection results regarding the rule-based approach and the disclosed method. Among the total 3,819 experimental Android apps, 603 of them are classified by the rule-based approach as AI Apps, while 1,366 are recognized as AI apps by the disclosed method, which is more than two times greater than that of the rule-based approach (242% more to be exact).

[0137] Fig. 7 shows a Venn diagram of detected AI apps in the rule-based approach and the disclosed method. Fig. 7 aims to illustrates the relationships between the outcomes of the disclosed method compared to the outcomes of the rule-based approach Android apps. In particular, Fig. 7 clearly shows that 546 Android apps, accounting for 90.55% (546/603) and 39.97% (546/1366) in the rule-based approach and the disclosed method, respectively, are both identified as AI Apps. The nine Android apps classified as AI by the rule-based approach were manually inspected and it was found that two apps were recognized as AI due to only one special tflite file being detected, while the remaining seven apps (App with the sha256 value: EC45FCB6288EDE61077FF80885F1A8EA99600E26B37E7DCABA7069C0134A4289) were classified as AI because the keywords, such as 'DecisionTreeClassifier' and 'KNeighborsClassifier', were detected in some descriptions provided in several JSON files, which are not convincing evidence to label these apps as AI.

[0138] Regarding the remaining 820 Android apps only classified by the disclosed method as AI, 86 Android apps (with a confidence level of 95% and margin of error of 10%) were

randomly selected and cross-validated by two inventors. Consensuses was reached if classification conflicts happen between the two inventors. In the process of cross-validation, it was found that some Android apps provide AI capabilities by utilizing public APIs offered by OpenAI. During the cross-validation, it was found that that some Android apps provide AI capabilities by utilizing OpenAI APIs, which are impossible for the rule-based method to detect due to the limited rules.

[0139] More specifically, some AI app developers invoked OpenAI APIs directly in Java code, while others implemented API invocations in the programming language C/C++ and compiled into a native dynamic library, in which the HTTPS request (https://api.openai.com/v1/chat/completions) is reflected in the extracted compiled Java class file or native library .so file. Customers subscribing to these apps will need to provide an OpenAI key to make full use of the functions given by these apps. In the end, it was confirmed that 83 of 86 AI apps classified by the disclosed method were AI Apps.

[0140] In summary, high precision (97.32%) and recall (91.01%), and the high inter-annotator reliability with regard to the AI candidate's identification reconfirm the effectiveness of the disclosed method and demonstrate its ability to extend to apps in the wild. Furthermore, the experimental result of detecting more than twice as many AI apps as the rule-based approach instantiates the outperformance of the disclosed method against the rule-based approach.

How useful is the AI summary generated by the disclosed method?

[0141] In terms of determining whether an Android app is AI-based or not, the disclosed method could effectively provide an accurate answer with a bunch of evidence. Here, the usefulness of the disclosed method in identifying detailed AI functionalities is evaluated provided by the determined AI-based apps.

[0142] To fulfil this purpose, six participants are first recruited. These six participants had expertise in Android app analysis with at least three years research experiences in Android. 20 Android apps are random selected and determined as AI apps in the previous section. For every selected app, an app description summary is generated via ChatGPT by feeding it reasoned AI candidates and provide the generated summary and the detailed AI candidates to participants. In the meanwhile, the app description is also retrieved from Google Play store and also offer it to the participants. More specifically, the participants are provided with two types of description

summary regarding each Android app. One is generated by ChatGPT with the help of the identified AI candidates, and the other is the description extracted from Google Play store.

[0143] To assess the quality of two different app description summaries,, participants are requested to evaluate the accuracy (i.e., Do you think that the summary is accurate concerning the AI functions the app provides?), completeness (i.e., Do you think that the summary is complete with regard to the provided AI tasks?), consistency (i.e., Do you think that the summary is concrete enough concerning the AI functions?), and recommendations for app users (i.e., Would you like to recommend the summary to other app users?) and developers (i.e., Would you like to recommend the summary to other app developers to inspire new app development?) by answering 5 linear scale questions with the rating scale spanning from 1 to 10. Lower rated scores indicate worse quality.

[0144] In total, 40 app descriptions were obtained for the selected 20 Android apps. To relieve the heavy burden of evaluating 40 app description summaries for every participant, the six participants were split into two groups. Each group of participants evaluated 10 Android apps and every app is evaluated by at least three different participants.

Summary Type	**Accuracy**		**Completeness**		**Consistency**		**Recom. App users**		**Recom. App devs.**	
	Avg.	Med.	Avg.	Med.	Avg.	Med.	Avg.	Med.	Avg.	Med.
AI Generated	8.67 ± 0.14	9	8.63 ± 0.13	9	8.55 ± 0.13	9	7.5 ± 0.31	8	8.57 ± 0.15	9
Developer offered	6.35 ± 0.45	8	6.17 ± 0.45	8	6.02 ± 0.45	8	6.87 ± 0.46	8	6.47 ± 0.45	8

Table 3: Average and median scale values voted for sampled Android apps by six participants. Recom. App users and Recom. App devs. corresponds to Recommendation for app users or app developers. Ave. and Med. correspond to the Average and Median value.

[0145] Table 3 shows the study results delineated by the average and median scale values. It is observed that the summary generated by the disclosed method is more than 2 greater than the value for the original description (i.e., developers' provided description) on most evaluation

aspects, while the median value is 1 point greater than the value for original app descriptions. Moreover, the standard error of the mean (SEM) of the summary generated by the disclosed method is approximately one-third of the SEM of the original app description, indicating that participants more consistently agree on the quality of the summary generated by the disclosed method.

[0146] Table 3 clearly shows that the summary generated by the disclosed method is conspicuously better than the description provided by their corresponding developers. In addition, it was found that the voted scores for both the summary generated by the disclosed method and the original app description are quite close on the metric of Recommendation for app users. Specifically, the median value is the same and the difference between the average scores is only 0.63, which reflects that different participants may have different preferences when recommending summaries to app users. In general, the description with higher accuracy, completeness, and consistency generated by the automated approach of the disclosed method is cherished by developers and researchers but it may not be valued by app users. However, some participants believe that app users would gain benefits from such descriptions as well.

[0147] In summary, the participant study with regard to the metrics of accuracy, completeness, consistency, recommendation for app users or app developers for both the AI (the disclosed method) generated summary and the app developers provided one shows that the summary generated by the disclosed method is much more preferred by the participants.

What insights can be gained from the analysis of the disclosed method?

Data Type	**Package & API**	**Model**	**HTTP Req.**	**Others**	**Total**
#. items	10,285	644	120	32	11,081

Table 4: Different types of identified AI candidates.

[0148] Here, the aim is to unveil the AI services underlying the identified AI apps. Specifically, the aim is to classify their specific domains and applications for AI candidates implemented in the identified AI apps in a previous section. Table 4 shows the number of the identified AI candidates in different self-explained categories. It clearly shows that the majority of the candidates (92.82%) are package names and APIs extracted from APK file.

[0149] The second biggest category is the model, which refers to the on-device AI model files, such as .tflite and .caffemodel. However, it is noted that AI models only play a small part among all AI candidates. Apart from package names, APIs, and AI models, it was found that some apps rely on the HTTPS requests (e.g., OpenAI APIs) to provide AI capabilities. For the remaining candidates which share few common patterns, these candidates are considered to be 'Others'.

[0150] With these identified AI candidates, a classification scheme can be defined which covers diverse applications and domains. As there are a significant number of AI candidates, it is impractical to perform the classification manually. Therefore, ChatGPT was utilised to classify AI domains and applications of each candidate.

[0151] Fig. 8 shows the distributions of classified domains and applications of the determined AI candidates. The area of the sector in different colour represents its proportion. On the top level, the AI services are classified into six common AI processes, including Computer Vision (54.80%), Machine Learning and Data Analysis (26.85%), Natural Language Processing (NLP) (13.38%), Others (2.75%), Audio and Speech Processing (1.90%), and Augmented Reality (0.32%), which is listed in the order of its proportion among the total number of AI candidates. Some of the candidates are given the domain name Others as it is hard for ChatGPT to determine the concrete AI process, such as some imported high-level package names.

[0152] It clearly represents that the majority of the AI functions are computer vision oriented. To be more specific, the detailed tasks in the second level in the domain of computer vision are Object Detection (45.97%), Text Recognition (17.95%), and Face Recognition (16.77%), etc. The second largest AI domain is Machine Learning and Data Analysis, which is occupied mostly by the detailed Data Processing (86.69%), such as the APIs invoked to do some tensor operations and input and output data parsing. The last and the second to last AI domain summarized in the AI candidates are Augmented Reality and Audio and Speech Processing, respectively.

[0153] Based on the classified AI candidate information, the domains and tasks are further categorised on the app level. For each application, the occurrence of domains and tasks associated with AI candidates are tallied. The application is then categorized under the domain and task that demonstrates the highest frequency. Table 5 shows the number of apps in different domains.

Domains	**Machine Learning and Data Analysis**	**Computer Vision**	**Natural Language Processing (NLP)**	**Audio and Speech Processing**	**Others**	**Augmented Reality**
#. Apps	643	643	43	30	14	2

Table 5: Number of Android apps in different domains.

[0154] The majority of AI applications fall into the Data Analysis domain, whereas most AI candidates are classified under Computer Vision. This discrepancy is primarily due to the overlap of candidates implemented across various apps, with a significant concentration in the Computer Vision category. Furthermore, the number of apps in both the Computer Vision and Data Analysis domains is nearly equal, placing them as the top two categories in both AI candidate classification and app categorization. In essence, the majority of AI candidates predominantly utilize machine learning and data processing algorithms to offer vision related features, aiding in everyday tasks like face and handwritten character recognition. Conversely, virtual reality functions are minimally incorporated in these apps. This distribution likely stems from the longstanding research and user demand for machine learning and vision-related functions. In contrast, virtual reality functions may lack compelling application scenarios and are not currently considered essential by most users.

[0155] In summary, the predominate commonly used AI services among Android apps and their underlying AI candidates are computer vision (54.80%) and data processing (26.85%) related as a result of popular daily tasks to fulfil for users via all sorts of mobile apps while the augmented reality-oriented ones are least provided due to the compelling application scenarios lacking despite it is a novel and popular concept nowadays.

Discussion

Implications

[0156] Security mitigation for software artifacts

[0157] Security patches are constantly merged into vulnerable software artifacts, such as the Android Operating System and the app involved third-party libraries, to prevent severe software compromising problems (e.g., confidential information leak) from happening. The results

described herein reveal that by asking LLM about the implementation basics, it could respond with accurate, at least to some extent, descriptions of whether they are AI-related or not. However, LLM can but is not limited to only responding to classification inquiries. With careful and heuristic prompt design, LLM can also be used to do code generation, bug fixes, etc. One possibility would be to use LLM as a neural knowledge to inquire if the API involves some existing vulnerabilities, for example, the vulnerability record in public Common Vulnerabilities and Exposures (CVE). Both researchers and developers could benefit from the vulnerability inquiries for LLM in that they could efficiently and effectively fix the potential vulnerabilities during their artifact's development. Consequently, more reliable and secure artifacts could be achieved.

[0158] Software artifacts analysis for researchers

[0159] The results described herein validate the potential of LLM in code summarization. The prevalence of cross-platform frameworks requires that code summarization should be taken on multiple programming languages. However, it is not straightforward to merge the summarizations on different programming languages due to the interactions between them. Detailed analysis of the interactions between different languages and careful design of prompts are necessary to make full use of the powerful potential offered by LLMs.

[0160] Advanced AI identification approaches and questionnaire with app developers

[0161] In the process of AI app identification, manual analysis reveal that it is hard to accurately identify any AI candidates for some self-claimed AI Android apps. More specifically, it was found that some Android apps claim that they are implemented with advanced AI functionalities and tasks in their app descriptions provided on the Google Play Store. The mismatch between the application description and AI implementation can hence be addressed by the disclosed method.

Conclusion

[0162] This disclosure provides an automated approach which leverages the advanced LLM to discovery AI candidates and pinpoint AI functionalities in Android apps. Experiments on sampled Android apps show that the disclosed method could have a better performance in AI app identification compared to rule-based approaches. The AI tasks and domains were then

characterized among the collected AI candidates. The results described herein revealed that the most popular AI service is computer vision related while the virtual reality-based functions are actually least provided even though it is one of the most popular concepts gaining most attentions both from AI researchers and early adopters nowadays. A user study was also conducted with respect to the usefulness of the disclosed method and the study results confirmed that the app summary generated by the disclosed method is much preferred by participants with regard to the metrics of accuracy, completeness, consistency, and recommendation for app developers.

[0163] It will be appreciated by persons skilled in the art that numerous variations and/or modifications may be made to the above-described embodiments, without departing from the broad general scope of the present disclosure. The present embodiments are, therefore, to be considered in all respects as illustrative and not restrictive.

CLAIMS:

1. A computer-implemented method for detecting artificial intelligence (AI) implementation in a software application comprising one or more application packages (APs), the method comprising:

extracting one or more application package (AP) strings from the software application, each of the one or more AP strings representing one of the one or more APs;

creating a prompt for a machine learning model, trained to generate output text, comprising the one or more AP strings, the prompt representing instructions to provide a classification of each of the one or more APs and provide functionality information of each of the one or more APs, the classification being AI relevant or non-AI relevant and the functionality information describing a functionality of the respective AP;

evaluating the machine learning model on the prompt to generate output text corresponding to the classification and the functionality information of each of the one or more APs; and

generating a report of the AI implementation in the software application based on the output text.

2. The method of claim 1, wherein creating the prompt for the machine learning model comprises creating a first prompt and a second prompt, wherein,

the first prompt represents instructions to provide the functionality information of each of the one or more APs; and

the second prompt represents instructions to provide the classification of each of the one or more APs.

3. The method of claim 2, wherein evaluating the machine learning model on the prompt comprises evaluating the machine learning model on the first prompt to generate initial output text and evaluating the machine learning model on the second prompt to generate the output text, wherein the second prompt is based on the initial output text.

4. The method of any one of the preceding claims, wherein the method further comprises analysing a description of the software application before extracting one or more AP strings from the software application to identify the one or more APs.

5. The method of claim 4, wherein analysing the description of the software application comprises performing a keyword search using terms relevant to AI.

6. The method of claim 4 or 5, wherein analysing a description of the software application comprises creating an initial prompt based on the description of the software application, the initial prompt representing instructions to provide a classification of the software application, the classification of the software application being likely to implement AI or unlikely to implement AI.

7. The method of any one of the preceding claims, wherein extracting the one or more AP strings from the software application comprises applying a static analysis method to the software application.

8. The method of any one of the preceding claims, wherein

the method further comprises creating a third prompt based on the output text, the third prompt representing instructions to generate the report; and

generating the report comprises evaluating the machine learning model on the third prompt to generate the report.

9. The method of claim 8, wherein the method further comprises classifying each of the one or more APs as an application programming interface (API) or a non-API and creating the third prompt further represents instructions to exclude APs classified as an API from the report.

10. The method of claim 8 or 9, wherein the first, second or third prompt comprises learning examples indicative of a desired output from evaluating the machine learning model on the first, second or third prompt, respectively.

11. The method of claim 10, wherein the learning examples are few-shot learning examples.

12. The method of any one of the preceding claims, wherein the method further comprises storing each AP string of APs classified as AI relevant and respective functionality information on an AI relevant database and storing each AP string of APs classified as non-AI relevant and respective functionality information on a non-AI relevant database.

13. The method of claim 12, wherein the method further comprises, after extracting one or more AP strings from the software application, accessing the AI relevant database and the non-AI relevant database to determine whether any one of the one or more AP strings are stored on either database.

14. The method of claim 13, wherein the method further comprises, upon determining that any one of the one or more AP strings are stored on either database, retrieving the functionality information from the respective database and excluding the respective AP from the prompt, wherein the report comprises the functionality information retrieved from the respective database.

15. The method of any one of the preceding claims, wherein the one or more APs are one or more of:

an API;

an asset file; and

a library.

16. The method any one of the preceding claims, wherein the method further comprises evaluating the machine learning model on a context prompt to provide context to the machine learning model to provide a desired output.

17. The method of any one of the preceding claims, wherein the method further comprises, upon classifying one of the one of more APs as AI relevant, generating and transmitting an alert to indicate detection of AI implementation in the software application to a user.

18. The method of any one of the preceding claims, wherein the software application is an Android application.

19. Software that, when executed by a computer, causes the computer to perform the method of any one of the preceding claims.

20. A system for detecting artificial intelligence (AI) implementation in a software application comprising one or more application packages (APs), the system comprising:

one or more processors configured to:

extract one or more application package (AP) strings from the software application, each of the one or more AP strings representing one of the one or more APs;

create a prompt for a machine learning model, trained to generate output text, comprising the one or more AP strings, the prompt representing instructions to provide a classification of each of the one or more APs and provide functionality information of each of the one or more APs, the classification being AI relevant or non-AI relevant and the functionality information describing a functionality of the respective AP;

evaluate the machine learning model on the prompt to generate output text corresponding to the classification and the functionality information of each of the one or more APs; and

generate a report of the AI implementation in the software application based on the output text.