MONITORING CHAIN-OF-THOUGHT IN REASONING MODELS TO MITIGATE REWARD HACKING

BACKGROUND

[0001] Generative response engines such as large language models represent a significant milestone in the field of artificial intelligence, revolutionizing computer-based natural language understanding and generation. Generative response engines, powered by advanced deep learning techniques, have demonstrated astonishing capabilities in tasks such as text generation, translation, summarization, and even code generation. Generative response engines can sift through vast amounts of text data, extract context, and provide coherent responses to a wide array of queries. Recently, generative response engines can perform a reasoning step to improve their ultimate response generation.

BRIEF DESCRIPTION OF THE SEVERAL VIEWS OF THE DRAWINGS

[0002] Details of one or more aspects of the subject matter described in this disclosure are set forth in the accompanying drawings and the description below. However, the accompanying drawings illustrate only some typical aspects of this disclosure and are therefore not to be considered limiting of its scope. Other features, aspects, and advantages will become apparent from the description, the drawings, and the claims.

[0003] FIG. 1 illustrates an example system supporting a generative response engine during inference operations in accordance with some embodiments of the present technology.

[0004] FIG. 2 is a block diagram illustrating an example computing environment that includes a second generative response engine monitoring the chain-of-thought from a first generative response engine in accordance with some embodiments of the present technology.

[0005] FIG. 3A, FIG. 3B, and FIG. 3C illustrate an example routine for monitoring the chain-of-thought from a first generative response engine by a second generative response engine in accordance with some embodiments of the present technology.

[0006] FIG. 4 is a block diagram illustrating an example machine-learning platform for implementing various aspects of this disclosure in accordance with some aspects of the present technology.

[0007] FIG. 5A, FIG. 5B, and FIG. 5C illustrates an example transformer architecture in accordance with some embodiments of the present technology.

[0008] FIG. 6 shows an example of a system for implementing some embodiments of the present technology.

DETAILED DESCRIPTION

[0009] Generative response engines such as large language models represent a significant milestone in the field of artificial intelligence, revolutionizing computer-based natural language understanding and generation. Generative response engines, powered by advanced deep learning techniques, have demonstrated astonishing capabilities in tasks such as text generation, translation, summarization, and even code generation.

[0010] Recent advancements in artificial intelligence (AI) have led to the development of increasingly capable generative response engines. However, these generative response engines sometimes exhibit a phenomenon known as reward hacking, where they exploit flaws or misspecifications in their reward functions to achieve high performance in unintended ways. This presents a fundamental challenge in ensuring that AI systems remain aligned with human intentions and operate safely in high-stakes applications. Reward hacking arises when generative response engines discover and exploit loopholes in their optimization criteria, leading to behavior that deviates from the intended objectives. This issue is particularly concerning as models become more sophisticated, making their reward exploitation strategies more difficult to detect. Traditional monitoring approaches, which rely on analyzing model outputs and behaviors, have proven insufficient in identifying instances of reward hacking, as the underlying reasoning processes of the models remain opaque. The present technology mitigates reward hacking by monitoring a first generative response engine's chain-of-thought (CoT) reasoning using a second generative response engine. This method leverages the internal reasoning steps output by the first generative response engine to detect patterns indicative of reward hacking, rather than solely relying on external outputs.

[0011] Studies have shown that a second generative response engine can effectively observe and analyze the CoT reasoning of a first generative response engine, identifying instances where the first generative response engine explicitly or implicitly attempts to exploit its reward function. By integrating CoT monitoring into AI oversight mechanisms, it becomes possible to intervene in real time to prevent reward hacking. This can be achieved by either modifying the behavior of the AI model or alerting human operators to potentially harmful reasoning patterns.

[0012] However, an additional challenge emerges when optimization pressure is applied to eliminate undesired reasoning patterns—AI models may learn to obfuscate their reasoning processes rather than eliminating harmful behaviors. As a result, excessive optimization of CoT reasoning may paradoxically lead to less transparent models, making oversight more difficult. Given these considerations, CoT monitoring represents a crucial technique for ensuring AI alignment, particularly in the context of highly capable generative models. Maintaining transparency in AI reasoning processes enables more effective oversight and intervention, reducing the risks associated with reward hacking.

[0013] FIG. 1 illustrates an example AI assistant service supporting a generative response engine during inference operations in accordance with some embodiments of the present technology. Although the example system depicts particular system components and an arrangement of such components, this depiction is to facilitate a discussion of the present technology and should not be considered limiting unless specified in the appended claims. For example, some components that are illustrated as separate can be combined with other components, and some components can be divided into separate components.

[0014] The generative response engine 110 is an artificial intelligence (AI) that can generate content in response to a prompt. The prompt can be from a human or a software entity (AI or applications). The prompt is generally in natural language but could be in code, including binary. Some examples of the generative response engine can include language models that generate language, such as CHATGPT, or other models, such as DALL-E, which generates images, and SORA, which generates videos. CHATGPT, DALL-E, and SORA are all provided by OPENAI, but the generative response engine is not limited to AI provided by OPENAI. The generative response engine can also be any type of generative AI and can include AI developed using various architectures such as diffusion models and transformers (e.g., autoregressive transformer architecture) and combinations of models.

[0015] In some instances, a language model, such as CHATGPT, can receive prompts to output images, video, code, applications, etc., which it can provide by interfacing with one or more other models, as will be addressed further herein.

[0016] Users and applications can interact with the generative response engine 110 through the front end 102. The front end 102 serves as the interface and intermediary between the user and the generative response engine. It encompasses the graphical user interface 104 and Application Programming Interfaces (APIs) 106 that facilitate communication, input

processing, and output presentation. Generally, users interact through a graphical user interface 104 that often includes a conversational interface, and applications interact through the API 106, but this is not a requirement.

[0017] The graphical user interface 104 is the platform through which users interact with the generative response engine 110. It can be a web-based chat window, a mobile application, or any interface that supports data input and output. The graphical user interface 104 facilitates a conversation between the user and the generative response engine, as the user provides prompts in the graphical user interface 104 to which the generative response engine responds and presents those responses in the graphical user interface 104. In some embodiments, graphical user interface 104 presents a conversational interface, which has attributes of a conversation thread between a user account and generative response engine 110.

[0018] The graphical user interface 104 is configured to perform input handling, context management, and output presentation. The type of inputs that can be received can be relative to the specifics of the generative response engine 110. For example, a language model is generally configured to accept text, but when the generative response engine is a multi-modal generative response engine, the front end 102 can accept voice and images/video.

[0019] In some embodiments, front end 102 can be an interface to accept any input types as part of the prompt, and downstream services can determine which generative response engine or collection of generative response engines are best suited to respond to the prompt.

[0020] The graphical user interface 104 is also configured to maintain the context of the conversation, which allows for coherent and relevant responses. For example, the graphical user interface 104 is responsible for providing the conversation thread and other relevant context accessible to the front end 102 to the generative response engine along with the specific prompt to the generative response engine. In an example, a conversation between the user account and the generative response engine 110 can have taken several turns (prompt, response, prompt, response, etc.). When the user account provides a further prompt, the graphical user interface 104 can provide that prompt to the generative response engine in the context of the entire conversation.

[0021] In another example, the graphical user interface 104 might be configured to provide a system prompt along with a user-provided prompt. A system prompt is hidden from the user account and is used to set the behavior and guidelines for the generative response engine. It can be used to define the AI's persona, style, and constraints. There can be levels of system

prompts. A highest level of a system prompt might be provided by the generative response engine 110 provider and its meant to establish policies for the behavior of generative response engine 110. This highest level os system prompt should is prohibited from being edited. A customization system prompt can be used to customize the behavior of the generative response engine and is often provided through an API call, or provided by a user account when creating a customized version of generative response engine 110. A still lower level of system prompt might include hidden information about a task. This can include chain-of-thought from a reasoning model, or context about an application the generative response engine 110 is working with to complete a task. Accordingly, the graphical user interface 104 does not always display all of the output of the generative response engine.

[0022] The graphical user interface 104 is also configured to display the responses from the generative response engine, which might include text, code snippets, images, or interactive elements.

[0023] In some embodiments, the generative response engine 110 can provide instructions to the front end 102 that instruct the graphical user interface 104 about how to display some of the output from the generative response engine. For example, the generative response engine can direct the graphical user interface 104 to present code in a code-specific format, or to present interactive graphics, or static images. In other examples, the generative response engine can direct the graphical user interface 104 to present an interactive document editor where the graphical user interface 104 can be presented with the document editor so that the user account and the generative response engine can collaborate on the document.

[0024] In some embodiments, the generative response engine 110 can provide instructions memory 126 to record facts in a personalization notepad, and front end 102 can be configured to notify the user account that a memory was created.

[0025] As noted above, the front end 102 can also provide one or more application programming interfaces (API(s)) 106. APIs enable developers to integrate the generative response engine's capabilities into external applications and services. They provide programmatic access to the generative response engine, allowing for customized interactions and functionalities. While APIs 106 are shown as part of a front end 102, this illustration takes the liberty of locating API 106 in front end 102 to refer to points of access to generative response engine 110 (i.e., graphical user interface 104 and APIs 106 are points of access and

generative response engine 110 sends and receives messages to them in similar ways). In reality, API 106 endpoints are located at context management service 120.

[0026] The APIs 106 can accept structured requests containing prompts, context, and configuration parameters. For example, an API can be used to provide prompts and divide the prompt into system prompts and user prompts. In some embodiments, the APIs 106 can provide specific inputs for which the generative response engine 110 is configured to respond with a specific behavior. For example, an API can be used to specify that it requires an output in a particular format or structured output. For example, in the chat completion API, the API call can specify parameters for the output, such as the max length for the desired output, and specify aspects of the tone of the language used in the response. Some common APIs are for participating in a conversation (Chat Completion API), for providing a single response (Completion API), for converting text into embeddings (Embeddings API), etc. The API can also be used to indicate specific decision boundaries that the generative response engine 110 might be trained to interpret. For example, the moderation API can take advantage of the AI assistant service 100's content moderation decision-making. In the case of the moderation API and others, the API might give access to services other than the generative response engine. For example, the moderation API might be an interface to moderation system 138, addressed below.

[0027] Some other common APIs include the Fine-Tuning API, which allows developers to customize models of the generative response engine using their own datasets; the Audio and Speech APIs, which cause the generative response engine to output speech or audio; and the Image Generation API, which causes the generative response engine to output images (which might require utilizing other models).

[0028] There can also be APIs that direct the generative response engine to interface with other applications or other generative AI engines. In such cases, the specific application or AI engine might be specified, or the generative response engine might be allowed to choose another application of AI engine to utilize in response to a prompt.

[0029] In short, the graphical user interface 104 and the APIs 106 can be used to provide prompts to the generative response engine. Prompts are sometimes differentiated into prompt types. For example, a system prompt can be a hidden prompt that sets the behavior and guidelines for the generative response engine. A user prompt is the explicit input provided by the user, which may include questions, commands, or information.

[0030] Sitting in between front end 102 and generative response engine 110 is a context management service 120. The function of context management service 120 is to manage and organize the flow of data among key subsystems, enabling the generative response engine 110 to generate responses that are contextually relevant, accurate, and enriched with additional information as required.

[0031] Action 122 facilitates auxiliary tasks that extend beyond basic text generation. In some embodiments, action 122 can be actions that correspond to an API 106. In some embodiments, action 122 can be agentic actions that the generative response engine 110 decides to take to carry out a user's intent as described in the prompt. For example, an action can be to call tool 130 or even another generative response engine 110.

[0032] Prompt 124 is the request or command provided by the user account through front end 102. In some embodiments, prompt 124 can be further supplemented by a system prompt and other information that might be included by graphical user interface 104 or API 106 or associated with a custom generative response engine 110. In some embodiments, prompt 124 can even be modified or enhanced by generative response engine 110 as addressed further below. Additionally, as the user account provides prompts and generative response engine 110 provides responses, a conversation thread forms. As the user account provides a new prompt, this is appended to the overall conversation and added to prompt 124. Thus, a user account might think of a first user-provided message as a first prompt and a second user-provided message as a second prompt, and so on, but prompt 124 as perceived by generative response engine 110 can include a thread of user-provided messages and responses from generative response engine 110 in a multi-turn conversation. The actors in the conversation thread can be labeled so that generative response engine 110 can review the turns of the conversation. Generally, prompt 124 will include an entire conversation thread, but in some instances, prompt 124 might need to be shortened if it exceeds a maximum accepted length (generally measured by a number of tokens).

[0033] Context management service 120 can also route prompts and response through moderation system 138. In some embodiments, prompts are provided to prompt safety system 134 before being provided to generative response engine 110. Prompt safety system 134 is configured to use one or more techniques to evaluate prompts to ensure a prompt is not requesting generative response engine 110 to generate moderated content. In some

embodiments, prompt safety system 134 can utilize text pattern matching, classifiers, and/or other AI techniques.

[0034] Since prompts can evolve over time through the course of a conversation, consisting of prompts and responses, prompts can be repeatedly evaluated at each turn in the conversation.

[0035] Memory 126 can facilitate continuity and personalization in conversations. It allows the system to maintain user-specific context, preferences, or details that may inform future interactions. A memory file can be persisted data from previous interactions or sessions that provide background information to maintain continuity. In some embodiments, memory can be recorded at the instruction of generative response engine 110 when generative response engine 110 identifies a fact or data that it determines should be saved in memory because it might be useful in later conversations or sessions. In some embodiments, memory 126 can also include synthesized concepts extracted from past conversation threads, and memory 126 can also encompass the ability of generative response engine 110 to search through past interactions to find relevant information to a current conversation thread.

[0036] Conversation metadata 128 can aggregate data points relevant to the conversation, including user prompt 124, action 122, and memory 126. This consolidated information package serves as the input for generative response engine 110. Conversation metadata 128 can label parts of a prompt as user provided, generative response engine provided, a system prompt, memory 126, data from action 122 or tool 130 (addressed below).

[0037] The generative response engine is the core engine that processes inputs (from context management service 120) and generates outputs. In some embodiments, the generative response engine is a generative transformer, or autoregressive transformer, but it could utilize other architectures.

[0038] A core feature of the generative response engine 110 is to generate content in response to prompts. The generative response engine 110 is configured to receive inputs from front end 102 that provide guidance on a desired output. The generative response engine can analyze the input and identify relevant patterns and associations in the data, and it has learned to generate a sequence of tokens that are predicted as the most likely continuation of the input. The generative response engine 110 generates responses by sampling from the probability distribution of possible tokens, guided by the patterns observed during its training. Two feature of the autoregressive transformer that result in this functionality is that the autoregressive transformer might use only the decoder part of the transformer architecture and that the it

utilizes self-attention. By utilizing the decoder part of the transformer architecture, the transformer focuses on predicting the next token in a sequence given the previous context. And the self-attention mechanism captures long-range dependencies in amongst tokens, allowing it to generate contextually relevant responses (in text, audio, images, and video).

[0039] In some embodiments, the generative response engine 110 can generate multiple possible responses before presenting the final one. The generative response engine 110 can generate multiple responses based on the input, and these responses are variations that the generative response engine 110 considers potentially relevant and coherent.

[0040] In some embodiments, the generative response engine 110 can evaluate generated responses based on certain criteria. These criteria can include relevance to the prompt, coherence, fluency, and sometimes adherence to specific guidelines or rules, depending on the application. Based on this evaluation, the generative response engine 110 can select the most appropriate response. This selection is typically the one that scores highest on the set criteria, balancing factors like relevance, informativeness, coherence, and content moderation instructions/training.

[0041] In some embodiments, an instruction provided by an API 106, a system prompt, or a decision made by generative response engine 110 can cause the generative response engine 110 to interpret a prompt and re-write it or improve the prompt for a desired purpose. For example, generative response engine 110 can determine to take a prompt to make a picture and enhance the prompt to yield a better picture. In these instances, generative response engine 110 can generate its own prompts, which can be provided to a tool 130 or provided to generative response engine 110 to yield a better output response than the original prompt might have.

[0042] The generative response engine 110 can also do more than generate content in response to a prompt. In some embodiments, the generative response engine 110 can utilize decision boundaries to determine the appropriate course of action based on the prompt. In some examples, a decision boundary might be used to cause the generative response engine to recognize that it is being asked to provide a response in a particular format such that it will generate its response constrained by the particular format. In some examples, a decision boundary can cause the model to refuse to generate a responsive output if the decision is that the responsive output would violate a moderation policy. In some examples, the decision boundary might cause the generative response engine to recognize that it needs to interface with another AI model or application to respond to the prompt. For example, when the

generative response engine is a language model, it might recognize that it is being asked to output an image, and therefore, it needs to interface with a model that can output images to provide a response to the prompt. In another example, the prompt might request a search of the Internet before responding. The generative response engine can use a decision boundary to recognize that it should conduct a search of the Internet and use the results of that search in responding to the prompt. In another example, the prompt might request that the generative response engine take an agentic action on behalf of the user by interacting with a third-party service (e.g., book a reservation for me at...), and the generative response engine can utilize a decision boundary to recognize that it needs to plan steps to locate the third-party service, contact the third-party service, and interact with the third-party service to complete the task and then report back to the user that the action has been completed.

[0043] When generative response engine 110 determines that it should take an agentic action on behalf of the user or it should call a tool to aid in providing a quality response to the user account, the generative response engine 110 might call a tool 130 or cause an action 122 to be performed. As indicated above, tools 130 can include internet browsers, editors such as code editors, other AI tools etc. Actions 122 are actions that the generative response engine 110 can cause to be performed, perhaps using tool 130. As used herein actions 122 should be considered to cover a broad array of actions that generative response engine 110 can perform with or without tools 130. Tools 130 are considered to cover a wide variety of services and software that encompass tools such as a computer operating system such that the generative response engine 110 can control the computer operating system on the user's behalf, to robotic actuators, to search browsers and specific applications.

[0044] Additionally, the generative response engine 110 can also generate portions of responses that are not displayed to the user. For example, the generative response engine 110 can direct the front end 102 to provide specific behaviors, such as directions for how to present the response from the generative response engine 110 to the user account. In another example, the generative response engine 110 can provide response portions dictated by an API, where portions of the response to the API might be for the consumption of the calling application but not for presentation to the end user. In another example, some generative response engine 110 are reasoning models, which are generative response engine 110 that are configured to output a raw chain-of-thought before preparing a final response to a prompt. The raw chain-of-thought might not be presented to a user account or application calling an API. Instead, another

generative response engine 110 might summarize the raw chain-of-thought into a more consumable and useful output for the user account or application.

[0045] In some embodiments, the output of generative response engine can be further analyzed by output safety system 136. While generative response engine 110 can perform some of its own moderation, there can be instances where it is desired to have another service review outputs for compliance with the moderation policy. The use of dashed lines in FIG. 1 differentiates a path using output safety system 136 and not using output safety system 136.

[0046] While FIG. 1 shows responses being provided back to front end 102 directly, in some embodiments, the responses might be returned by way of context management service 120.

[0047] FIG. 2 is a block diagram illustrating an example computing environment that includes a second generative response engine monitoring the chain-of-thought from a first generative response engine in accordance with some embodiments of the present technology.

[0048] In some examples, the first generative response engine 202 includes a context window 206, which stores or references a chain-of-thought 208. The chain-of-thought 208 represents a sequence of internal reasoning steps generated by the first generative response engine 202 when performing a given task.

[0049] As shown, the chain-of-thought 208 is streamed or otherwise provided to the second generative response engine 204, which is configured to monitor the chain-of-thought 208 for indicators of reward hacking or other undesirable reasoning patterns. In some examples, the second generative response engine 204 can modify the operation of the first generative response engine 202 based on its observations as addressed in greater detail herein.

[0050] The arrow from the chain-of-thought 208 to the second generative response engine 204 indicates the flow of chain-of-thought data, while the arrow returning from the second generative response engine 204 to the first generative response engine 202 indicates the second generative response engine's ability to provide instructions or modifications to the first generative response engine 202.

[0051] FIG. 3A illustrates an example routine for monitoring the chain-of-thought from a first generative response engine by a second generative response engine in accordance with some embodiments of the present technology. Although the example routine depicts a particular sequence of operations, the sequence may be altered without departing from the scope of the present disclosure. For example, some of the operations depicted may be performed in parallel or in a different sequence that does not materially affect the function of the routine. In other

examples, different components of an example device or system that implements the routine may perform functions at substantially the same time or in a specific sequence.

[0052] According to some examples, the method includes generating a chain-of-thought while performing a task defined by a task specification at block 302. For example, the first generative response engine 202 illustrated in FIG. 2 may generate a chain-of-thought while performing a task defined by a task specification. In some examples, the chain-of-thought is a plurality of reasoning steps is used by the first generative response engine to guide inference operations that lead to the final output. The chain-of-thought is often not directly visible in a final output of the generative response engine. For example, the chain-of-thought from the first generative response engine 202 is often summarized and the resulting summary is often provided with the final output.

[0053] According to some examples, the method includes monitoring the chain-of-thought from the first generative response engine at block 304. For example, the second generative response engine 204 illustrated in FIG. 2 may monitor the chain-of-thought from the first generative response engine. For example, the second generative response engine is configured as an oversight agent.

[0054] In some embodiments, the chain-of-thought is streamed to the second generative response engine as it is generated by the first generative response engine. In some embodiments, the chain-of-thought is copied from the memory (context window 206) of the first generative response engine 202.

[0055] According to some examples, the method includes identifying at least one indicator of reward hacking in the chain-of-thought at block 306. For example, the second generative response engine 204 illustrated in FIG. 2 may identify at least one indicator of reward hacking in the chain-of-thought. In some examples, the at least one indicator of reward hacking comprises a reasoning step or pattern that corresponds to the first generative response engine exploiting a reward function in a manner not intended by the task specification. In some embodiments, second generative response engine 204 can be specifically trained (fine-tuned) to perform this task.

[0056] According to some examples, the method includes modifying the operation of the first generative response engine to mitigate the reward hacking at block 308. For example, the second generative response engine 204 illustrated in FIG. 2 may modify the operation of the first generative response engine to mitigate the reward hacking.

[0057] There are several mechanisms by which the second generative response engine 204 can mitigate the reward hacking by the first generative response engine 202. These are addresssed with respect to FIG. 3B, FIG. 3C, and in block 312, block 314, and block 316.

[0058] For example, one method of mitigating the reward hacking includes outputting an alert to a human operator describing the at least one indicator of reward hacking in the chain-of-thought at block 312. For example, the second generative response engine 204 illustrated in FIG. 2 may output an alert to a human operator describing the at least one indicator of reward hacking in the chain-of-thought.

[0059] For example, another method of mitigating the reward hacking includes adjusting a reward signal provided to the first AI model at block 314. For example, the second generative response engine 204 illustrated in FIG. 2 may adjusting a reward signal provided to the first AI model.

[0060] For example, another method of mitigating the reward hacking includes overriding a planned action of the first generative response engine at block 316. For example, the second generative response engine 204 illustrated in FIG. 2 may overriding a planned action of the first generative response engine.

[0061] Another example of mitigating the reward hacking is illustrated in FIG. 3B. According to some examples, the method includes revising the chain-of-thought to eliminate the at least one indicator of reward hacking in the chain-of-thought at block 310. For example, the second generative response engine 204 illustrated in FIG. 2 may revise the chain-of-thought to eliminate the at least one indicator of reward hacking in the chain-of-thought. The method also includes replacing the chain-of-thought with the revised chain-of-thought in a context window of the second generative response engine at block 318. For example, the second generative response engine 204 illustrated in FIG. 2 may replace the chain-of-thought with the revised chain-of-thought in a context window of the second generative response engine. The second generative response engine processes the task based on the revised chain-of-thought.

[0062] Another example of mitigating the reward hacking is illustrated in FIG. 3C. According to some examples, the method includes causing the first generative response engine to terminate performance of the task at block 320. For example, the second generative response engine 204 illustrated in FIG. 2 may cause the first generative response engine to terminate

performance of the task. The method further includes revising the task specification to be less susceptible to exploiting the reward function, which yields a revised task specification defining the task at block 322. For example, the second generative response engine 204 illustrated in FIG. 2 may revise the task specification to be less susceptible to exploiting the reward function, which yields a revised task specification defining the task. Additonally, the method includes providing the revised task specification to the first generative response engine to perform the task again at block 324. For example, the second generative response engine 204 illustrated in FIG. 2 may providing the revised task specification to the first generative response engine to perform the task again.

[0063] FIG. 4 is a block diagram illustrating an example machine learning platform for implementing various aspects of this disclosure in accordance with some aspects of the present technology. Although the example system depicts particular system components and an arrangement of such components, this depiction is to facilitate a discussion of the present technology and should not be considered limiting unless specified in the appended claims. For example, some components that are illustrated as separate can be combined with other components, and some components can be divided into separate components.

[0064] System 400 may include data input engine 410 that can further include data retrieval engine 412 and data transform engine 414. Data retrieval engine 412 may be configured to access, interpret, request, or receive data, which may be adjusted, reformatted, or changed (e.g., to be interpretable by another engine, such as data input engine 410). For example, data retrieval engine 412 may request data from a remote source using an API. Data input engine 410 may be configured to access, interpret, request, format, re-format, or receive input data from data sources(s) 401. For example, data input engine 410 may be configured to use data transform engine 414 to execute a re-configuration or other change to data, such as a data dimension reduction. In some embodiments, data sources(s) 401 may be associated with a single entity (e.g., organization) or with multiple entities. Data sources(s) 401 may include one or more of training data 402a (e.g., input data to feed a machine learning model as part of one or more training processes), validation data 402b (e.g., data against which at least one processor may compare model output with, such as to determine model output quality), and/or reference data 402c. In some embodiments, data input engine 410 can be implemented using at least one computing device. For example, data from data sources(s) 401 can be obtained through one or more I/O devices and/or network interfaces. Further, the data may be stored

(e.g., during execution of one or more operations) in a suitable storage or system memory. Data input engine 410 may also be configured to interact with a data storage, which may be implemented on a computing device that stores data in storage or system memory.

[0065] System 400 may include featurization engine 420. Featurization engine 420 may include feature annotating & labeling engine 422 (e.g., configured to annotate or label features from a model or data, which may be extracted by feature extraction engine 424), feature extraction engine 424 (e.g., configured to extract one or more features from a model or data), and/or feature scaling & selection engine 426 Feature scaling & selection engine 426 may be configured to determine, select, limit, constrain, concatenate, or define features (e.g., AI features) for use with AI models.

[0066] System 400 may also include machine learning (ML) ML modeling engine 430, which may be configured to execute one or more operations on a machine learning model (e.g., model training, model re-configuration, model validation, model testing), such as those described in the processes described herein. For example, ML modeling engine 430 may execute an operation to train a machine learning model, such as adding, removing, or modifying a model parameter. Training of a machine learning model may be supervised, semi-supervised, or unsupervised. In some embodiments, training of a machine learning model may include multiple epochs, or passes of data (e.g., training data 402a) through a machine learning model process (e.g., a training process). In some embodiments, different epochs may have different degrees of supervision (e.g., supervised, semi-supervised, or unsupervised). Data into a model to train the model may include input data (e.g., as described above) and/or data previously output from a model (e.g., forming a recursive learning feedback). A model parameter may include one or more of a seed value, a model node, a model layer, an algorithm, a function, a model connection (e.g., between other model parameters or between models), a model constraint, or any other digital component influencing the output of a model. A model connection may include or represent a relationship between model parameters and/or models, which may be dependent or interdependent, hierarchical, and/or static or dynamic. The combination and configuration of the model parameters and relationships between model parameters discussed herein are cognitively infeasible for the human mind to maintain or use. Without limiting the disclosed embodiments in any way, a machine learning model may include millions, billions, or even trillions of model parameters. ML modeling engine 430 may include model selector engine 432 (e.g., configured to select a model from among a plurality of

models, such as based on input data), parameter engine 434 (e.g., configured to add, remove, and/or change one or more parameters of a model), and/or model generation engine 436 (e.g., configured to generate one or more machine learning models, such as according to model input data, model output data, comparison data, and/or validation data).

[0067] In some embodiments, model selector engine 432 may be configured to receive input and/or transmit output to ML algorithms database 470. Similarly, featurization engine 420 can utilize storage or system memory for storing data and can utilize one or more I/O devices or network interfaces for transmitting or receiving data. ML algorithms database 470 may store one or more machine learning models, any of which may be fully trained, partially trained, or untrained. A machine learning model may be or include, without limitation, one or more of (e.g., such as in the case of a metamodel) a statistical model, an algorithm, a neural network (NN), a convolutional neural network (CNN), a generative neural network (GNN), a Word2Vec model, a bag of words model, a term frequency-inverse document frequency (tf-idf) model, a GPT (Generative Pre-trained Transformer) model (or other autoregressive model), a diffusion model, a diffusion-transformer model, an encoder such as BERT (Bidirectional Encoder Representations from Transformers) or LXMERT (Learning Cross-Modality Encoder Representations from Transformers), a Proximal Policy Optimization (PPO) model, a nearest neighbor model (e.g., k nearest neighbor model), a linear regression model, a k-means clustering model, a Q-Learning model, a Temporal Difference (TD) model, a Deep Adversarial Network model, or any other type of model described further herein. Some of the ML algorithms in ML algorithms database 470 can be considered generative response engines. Generative response engines are those models are commonly referred to as Generative AI, and that can receive an input prompt and generate additional content based on the prompt. GPTs, diffusion models, and diffusion-transformer models are some non-limiting examples of generative response engines. Some specific examples of generative response engines that can be stored in the ML algorithms database 470 include versions DALL·E, CHAT GPT, and SORA, all provided by OPEN AI.

[0068] System 400 can further include predictive output generation engine 445 and output validation engine 450 (e.g., configured to apply validation data to machine learning model output). Predictive output generation engine 445 can analyze the input and identify relevant patterns and associations in the data it has learned to generate a sequence of words that predictive output generation engine 445 predicts is the most likely continuation of the input

using one or more models from the ML algorithms database 470, aiming to provide a coherent and contextually relevant answer. Predictive output generation engine 445 generates responses by sampling from the probability distribution of possible words and sequences, guided by the patterns observed during its training. In some embodiments, predictive output generation engine 445 can generate multiple possible responses before presenting the final one. Predictive output generation engine 445 can generate multiple responses based on the input, and these responses are variations that predictive output generation engine 445 considers potentially relevant and coherent. Output validation engine 450 can evaluate these generated responses based on certain criteria. These criteria can include relevance to the prompt, coherence, fluency, and sometimes adherence to specific guidelines or rules, depending on the application. Based on this evaluation, output validation engine 450 selects the most appropriate response. This selection is typically the one that scores highest on the set criteria, balancing factors like relevance, informativeness, and coherence.

[0069] System 400 can further include feedback engine 460 (e.g., configured to apply feedback from a user and/or machine to a model) and model refinement engine 455 (e.g., configured to update or re-configure a model). In some embodiments, feedback engine 460 may receive input and/or transmit output (e.g., output from a trained, partially trained, or untrained model) to outcome metrics database 465. Outcome metrics database 465 may be configured to store output from one or more models and may also be configured to associate output with one or more models. In some embodiments, outcome metrics database 465, or other device (e.g., model refinement engine 455 or feedback engine 460), may be configured to correlate output, detect trends in output data, and/or infer a change to input or model parameters to cause a particular model output or type of model output. In some embodiments, model refinement engine 455 may receive output from predictive output generation engine 445 or output validation engine 450. In some embodiments, model refinement engine 455 may transmit the received output to featurization engine 420 or ML modeling engine 430 in one or more iterative cycles.

[0070] The engines of system 400 may be packaged functional hardware units designed for use with other components or a part of a program that performs a particular function (e.g., of related functions). Any or each of these modules may be implemented using a computing device. In some embodiments, the functionality of system 400 may be split across multiple computing devices to allow for distributed processing of the data, which may improve output

speed and reduce computational load on individual devices. In some embodiments, system 400 may use load-balancing to maintain stable resource load (e.g., processing load, memory load, or bandwidth load) across multiple computing devices and to reduce the risk of a computing device or connection becoming overloaded. In these or other embodiments, the different components may communicate over one or more I/O devices and/or network interfaces.

[0071] System 400 can be related to different domains or fields of use. Descriptions of embodiments related to specific domains, such as natural language processing or language modeling, is not intended to limit the disclosed embodiments to those specific domains, and embodiments consistent with the present disclosure can apply to any domain that utilizes predictive modeling based on available data.

[0072] FIG. 5A, FIG. 5B, and FIG. 5C illustrates an example transformer architecture in accordance with some embodiments of the present technology. Examples of ML models that use a transformer neural network (e.g., transformer architecture 500) can include, e.g., generative pretrained transformer (GPT) models and Bidirectional Encoder Representations from Transformer (BERT) models. The transformer architecture 500, which is illustrated in FIG. 5A, FIG. 5B, and FIG. 5C, includes inputs 502, input embedding block 504, positional encodings 506, encoder 508 including encode blocks 510, decoder 512 including decode blocks 514, linear block 516, softmax block 518, and output probabilities 520.

[0073] Input embedding block 504 is used to provide representations for words. For example, embedding can be used in text analysis. According to certain non-limiting examples, the representation is a real-valued vector that encodes the meaning of the word in such a way that words that are closer in the vector space are expected to be similar in meaning. Word embeddings can be obtained using language modeling and feature learning techniques, where words or phrases from the vocabulary are mapped to vectors of real numbers. According to certain non-limiting examples, the input embedding block 504 can be learned embeddings to convert the input tokens and output tokens to vectors of dimension that have the same dimension as the positional encodings, for example.

[0074] Positional encodings 506 provide information about the relative or absolute position of the tokens in the sequence. According to certain non-limiting examples, positional encodings 506 can be provided by adding positional encodings to the input embeddings at the inputs to the encoder 508 and decoder 512. The positional encodings have the same dimension as the embeddings, thereby enabling a summing of the embeddings with the positional encodings.

There are several ways to realize the positional encodings, including learned and fixed. For example, sine and cosine functions having different frequencies can be used. That is, each dimension of the positional encoding corresponds to a sinusoid. Other techniques of conveying positional information can also be used, as would be understood by a person of ordinary skill in the art. For example, learned positional embeddings can instead be used to obtain similar results. An advantage of using sinusoidal positional encodings rather than learned positional encodings is that doing so allows the model to extrapolate to sequence lengths longer than the ones encountered during training.

[0075] Encoder 508 can use stacked self-attention and point-wise, fully connected layers. Encoder 508 can be a stack of N identical layers (e.g., N = 6), and each layer can be an encode block, as illustrated by encode block 510 shown in FIG. 5B. Each encode block 510 has two sub-layers: (i) a first sub-layer has a multi-head attention block 522 and (ii) a second sub-layer has a feed forward block 526, which can be a position-wise fully connected feed-forward network. The feed forward block 526 can use a rectified linear unit (ReLU).

[0076] Encoder 508 uses a residual connection around each of the two sub-layers, followed by an add & norm block 524, which performs normalization. For example, the output of each sub-layer can be LayerNorm(x+ Sublayer(x)). To facilitate these residual connections, all sub-layers in the model, as well as the embedding layers, produce output data having a same dimension.

[0077] Similar to encoder 508, decoder 512 uses stacked self-attention and point-wise, fully connected layers. Decoder 512 can also be a stack of M identical layers (e.g., M = 6), and each layer can be a decode block, as illustrated by decode block 512 shown in FIG. 5B. In addition to the two sub-layers (i.e., the sublayer with multi-head attention block 522 and the sub-layer with feed forward block 526) found in encode block 510, decode block 514 can include a third sub-layer, which performs multi-head attention over the output of the encoder stack. Similar to encoder 508, decoder 512 uses residual connections around each of the sub-layers, followed by layer normalization. Additionally, the sub-layer with multi-head attention block 522 can be modified in the decoder stack to prevent positions from attending to subsequent positions. This masking, combined with the fact that the output embeddings are offset by one position, can ensure that the predictions for position i can depend only on the known output data at positions less than i.

[0078] Linear block 516 can be a learned linear transformation. For example, when transformer architecture 500 is being used to translate from a first language into a second language, linear block 516 can project the output from the last decode softmax block 518 into word scores for the second language (e.g., a score value for each unique word in the target vocabulary) at each position in the sentence. For instance, if the output sentence has seven words and the provided vocabulary for the second language has 10,000 unique words, then 10,000 score values are generated for each of those seven words. The score values indicate the likelihood of occurrence for each word in the vocabulary in that position of the sentence.

[0079] Softmax block 518 then turns the scores from linear block 516 into output probabilities 520 (which add up to 1.0). In each position, the index provides for the word with the highest probability, and then maps that index to the corresponding word in the vocabulary. Those words then form the output sequence of transformer architecture 500. The softmax operation is applied to the output from linear block 516 to convert the raw numbers into output probabilities 520 (e.g., token probabilities).

[0080] FIG. 6 shows an example of computing system 600, which can be, For example, any computing device making up any engine illustrated in FIG. 1 or any component thereof.

[0081] In some embodiments, computing system 600 is a single device, or a distributed system in which the functions described in this disclosure can be distributed within a datacenter, multiple data centers, a peer network, etc. In some embodiments, one or more of the described system components represents many such components each performing some or all of the function for which the component is described. In some embodiments, the components can be physical or virtual devices.

[0082] In some embodiments, computing system 600 may comprise one or more computing resources provisioned from a "cloud computing" provider, For example, AMAZON ELASTIC COMPUTE CLOUD ("AMAZON EC2"), provided by AMAZON, INC. of Seattle, Washington; SUN CLOUD COMPUTER UTILITY, provided by SUN MICROSYSTEMS, INC. of Santa Clara, California; AZURE, provided by MICROSOFT CORPORATION of Redmond, Washington, GOOGLE CLOUD PLATFORM, provided by ALPHABET, INC. of Mountain View, California, and the like.

[0083] Example computing system 600 includes at least one processing unit (CPU or processor) 604 and connection 602 that couples various system components including system memory 608, such as read-only memory (ROM) 610 and random access memory (RAM) 612 to

processor 604. Memory 608 can be a volatile or non-volatile memory device, and can be a hard disk or other types of non-transitory computer readable media which can store data that are accessible by a computer, such as magnetic cassettes, flash memory cards, solid state memory devices, digital versatile disks, cartridges, random access memories (RAMs), read-only memory (ROM), and/or some combination of these devices.

[0084] Memory 608 can include software services, servers, logic, etc., that when the code that defines such software is executed by the processor 604, it causes the system to perform a function. In some embodiments, a hardware service that performs a particular function can include the software component stored in a computer-readable medium in connection with the necessary hardware components, such as processor 604, connection 602, output device 622, etc., to carry out the function.

[0085] Computing system 600 can include a cache of high-speed memory 606 connected directly with, in close proximity to, or integrated as part of processor 604.

[0086] Connection 602 can be a physical connection via a bus, or a direct connection into processor 604, such as in a chipset architecture. Connection 602 can also be a virtual connection, networked connection, or logical connection.

[0087] Processor 604 can include any general purpose processor and a hardware service or software service stored in memory 608, configured to control processor 604 as well as a special-purpose processor where software instructions are incorporated into the actual processor design. Processor 604 may essentially be a completely self-contained computing system, containing multiple cores or processors, a bus, memory controller, cache, etc. A multi-core processor may be symmetric or asymmetric. Processor 604 can be physcial or virtual.

[0088] To enable user interaction, computing system 600 includes an input device 626, which can represent any number of input mechanisms, such as a microphone for speech, a touch-sensitive screen for gesture or graphical input, keyboard, mouse, motion input, speech, etc. Computing system 600 can also include output device 622, which can be one or more of a number of output mechanisms known to those of skill in the art. In some instances, multimodal systems can enable a user to provide multiple types of input/output to communicate with computing system 600. Computing system 600 can include communication interface 624, which can generally govern and manage the user input and system output. There is no restriction on operating on any particular hardware arrangement, and therefore the basic

features here may easily be substituted for improved hardware or firmware arrangements as they are developed.

[0089] In some embodiments, computing system 600 can refer to a combination of a personal computing device interacting with components hosted in a data center, where both the computing device and the components in the data center. In such examples, both the personal computing device and the components in the datacenter might have a processor, cache, memory, storage, etc.

[0090] For clarity of explanation, in some instances, the present technology may be presented as including individual functional blocks including functional blocks comprising devices, device components, steps or routines in a method embodied in software, or combinations of hardware and software.

[0091] Any of the steps, operations, functions, or processes described herein may be performed or implemented by a combination of hardware and software services or services, alone or in combination with other devices. In some embodiments, a service can be software that resides in memory of a client device and/or one or more servers of a content management system and perform one or more functions when a processor executes the software associated with the service. In some embodiments, a service is a program or a collection of programs that carry out a specific function. In some embodiments, a service can be considered a server. The memory can be a non-transitory computer-readable medium.

[0092] In some embodiments, the computer-readable storage devices, mediums, and memories can include a cable or wireless signal containing a bit stream and the like. However, when mentioned, non-transitory computer-readable storage media expressly exclude media such as energy, carrier signals, electromagnetic waves, and signals per se.

[0093] Methods according to the above-described examples can be implemented using computer-executable instructions that are stored or otherwise available from computer-readable media. Such instructions can comprise, For example, instructions and data which cause or otherwise configure a general purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions. Portions of computer resources used can be accessible over a network. The executable computer instructions may be, For example, binaries, intermediate format instructions such as assembly language, firmware, or source code. Examples of computer-readable media that may be used to store instructions, information used, and/or information created during methods according to described examples

include magnetic or optical disks, solid-state memory devices, flash memory, USB devices provided with non-volatile memory, networked storage devices, and so on.

[0094] Devices implementing methods according to these disclosures can comprise hardware, firmware and/or software, and can take any of a variety of form factors. Typical examples of such form factors include servers, laptops, smartphones, small form factor personal computers, personal digital assistants, and so on. The functionality described herein also can be embodied in peripherals or add-in cards. Such functionality can also be implemented on a circuit board among different chips or different processes executing in a single device, by way of further example.

[0095] The instructions, media for conveying such instructions, computing resources for executing them, and other structures for supporting such computing resources are means for providing the functions described in these disclosures.

[0096] Aspects:

[0097] The present technology includes computer-readable storage mediums for storing instructions, and systems for executing any one of the methods embodied in the instructions addressed in the aspects of the present technology presented below: