



(51) International Patent Classification:

G06F 16/33 (2019.01) G06F 40/40 (2020.01)
G06F 40/284 (2020.01) G06F 40/00 (2020.01)

(21) International Application Number:

PCT/US2024/022528

(22) International Filing Date:

01 April 2024 (01.04.2024)

(25) Filing Language:

English

(26) Publication Language:

English

(30) Priority Data:

63/456,300 31 March 2023 (31.03.2023) US

(71) Applicant: **GOOGLE LLC** [US/US]; 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(72) Inventors: **DAI, Andrew Mingbo**; c/o Google LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **LIU, Ruibo**; c/o Google LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **CHU, Eric**; c/o Google LLC, 1600 Amphitheatre Parkway, Moun-

tain View, California 94043 (US). **ZHOU, Dengyong**; c/o Google LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US). **WEI, Jason Weng**; c/o Google LLC, 1600 Amphitheatre Parkway, Mountain View, California 94043 (US).

(74) Agent: **PROBST, Joseph J.** et al.; DORITY & MAN-
NING, P.A., P.O. BOX 1449, GREENVILLE, South Carolina 29602 (US).

(81) Designated States (unless otherwise indicated, for every kind of national protection available): AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BN, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CV, CZ, DE, DJ, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IQ, IR, IS, IT, JM, JO, JP, KE, KG, KH, KN, KP, KR, KW, KZ, LA, LC, LK, LR, LS, LU, LY, MA, MD, MG, MK, MN, MU, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PA, PE, PG, PH, PL, PT, QA, RO, RS, RU, RW, SA, SC, SD, SE, SG, SK, SL, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, WS, ZA, ZM, ZW.

(54) Title: EFFICIENT USE OF TOOLS BY LANGUAGE MODELS

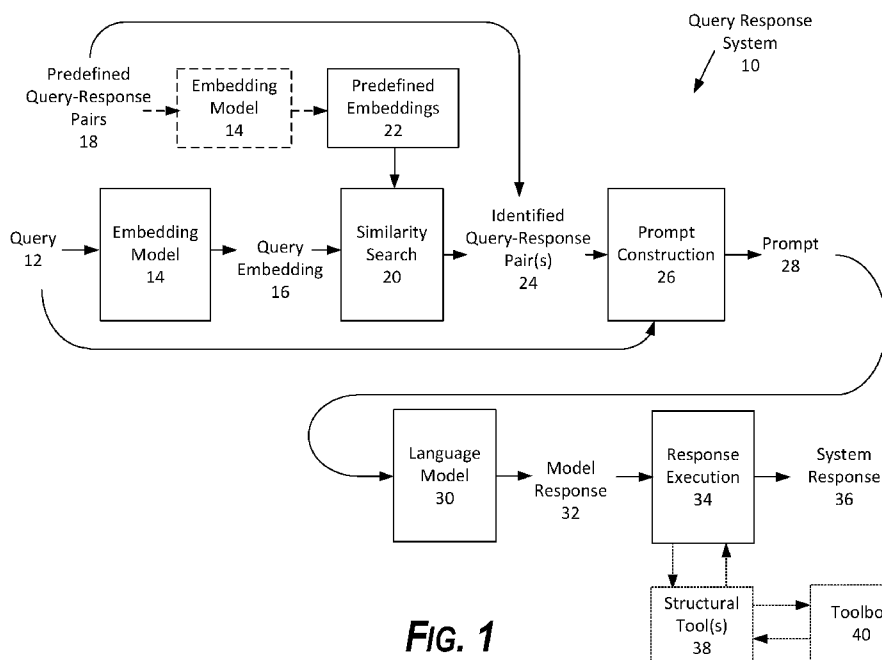


FIG. 1

(57) Abstract: Provided are computer systems that enable machine-learned language models to choose the correct structural tools to leverage when solving challenging tasks. In particular, current language models rely heavily on internal knowledge to solve all downstream tasks, which causes hallucination and ungrounded answers. In contrast, proposed herein is a new tool-using framework for language models, which enables the language models to smartly route a query to the most relevant pre-defined skills.

(84) **Designated States** (*unless otherwise indicated, for every kind of regional protection available*): ARIPO (BW, CV, GH, GM, KE, LR, LS, MW, MZ, NA, RW, SC, SD, SL, ST, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, ME, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, KM, ML, MR, NE, SN, TD, TG).

Declarations under Rule 4.17:

— *of inventorship (Rule 4.17(iv))*

Published:

— *with international search report (Art. 21(3))*
— *before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments (Rule 48.2(h))*

EFFICIENT USE OF TOOLS BY LANGUAGE MODELS

RELATED APPLICATIONS

[0001] This application claims priority to and the benefit of United States Provisional Patent Application Number 63/456,300, filed March 31, 2023. United States Provisional Patent Application Number 63/456,300 is hereby incorporated by reference in its entirety.

FIELD

[0002] The present disclosure relates generally to the use of machine learning for language modeling. More particularly, the present disclosure relates to a computing system that generates prompts that enable machine-learned language models to efficiently use structural tools to generate a response for a query.

BACKGROUND

[0003] Natural language processing (NLP) has seen rapid advances in recent years and such advances are primarily attributable to improvements in learning based algorithms and other aspects of machine or “neural” learning. One particular task within the field of NLP is contextual text generation. In the contextual text generation task, an agent (e.g., a machine learning model) is tasked with generating output text from a given context. In some example approaches to the contextual text generation task, a text-to-text model reads the input contextual text and then directly produces the output text. One example of a contextual text generation task is a question answering task in which the input context contains a question and the desired output text is the answer to the question.

[0004] Current state-of-the-art models for contextual text generation tend to be Transformer-based neural models, several of which are generally referred to as “large language models” (or LLMs). However, these so-called large language models suffer from a number of drawbacks. Specifically, while pre-trained large language models display significant intelligence, their knowledge is constrained to the information contained in (and learned from) their training datasets and/or information introduced within the contextual text input. Thus, their knowledge of factual information is severely limited and generally frozen in time. As such, when requested to produce an output that contains factual information, the models typically either hallucinate incorrect facts or supply outdated information. Reliance upon incorrect factual information can result in inefficiencies in which incorrect actions (e.g.,

computerized actions) are taken and need to be corrected or otherwise remediated, resulting in redundant and unnecessary use of resources (e.g., computing resources).

[0005] The inability of current language models to access external knowledge is just one example of a larger challenge associated with such models. Namely, existing language models are largely unable to select, access, or leverage various structural tools such as computational tools (e.g., calculators), information retrieval tools (e.g., stored data, search engines, knowledge graphs), and/or programming tools (e.g., compilers, programming language interpreters, etc.). Access to these tools would greatly improve the ability of language models to handle far more diverse and complex queries.

SUMMARY

[0006] Aspects and advantages of embodiments of the present disclosure will be set forth in part in the following description, or can be learned from the description, or can be learned through practice of the embodiments.

[0007] One example aspect of the present disclosure is directed to a computer-implemented method to perform contextual text generation. The method includes obtaining, by a computing system comprising one or more computing devices, data descriptive of a query. The method includes generating, by the computing system, a query embedding for the query, wherein the query embedding is expressed within a latent embedding space. The method includes performing, by the computing system, a similarity search for the query embedding within the latent embedding space to identify one or more previously-defined embeddings associated with one or more previously-defined query-response pairs. The method includes generating, by the computing system, a prompt based on the query and the one or more previously-defined query-response pairs. The method includes providing, by the computing system, the prompt as an input for processing by a machine-learned language model. The method includes receiving, by the computing system, a model response to the query that was output by the machine-learned language model based on processing of the prompt.

[0008] Other aspects of the present disclosure are directed to various systems, apparatuses, non-transitory computer-readable media, user interfaces, and electronic devices.

[0009] These and other features, aspects, and advantages of various embodiments of the present disclosure will become better understood with reference to the following description and appended claims. The accompanying drawings, which are incorporated in and constitute

a part of this specification, illustrate example embodiments of the present disclosure and, together with the description, serve to explain the related principles.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] Detailed discussion of embodiments directed to one of ordinary skill in the art is set forth in the specification, which makes reference to the appended figures, in which:

[0011] Figure 1 depicts a block diagram of an example data flow to generate a system response to a query according to example embodiments of the present disclosure.

[0012] Figure 2 depicts a graphical diagram of an example process to generate a system response to a query according to example embodiments of the present disclosure.

[0013] Figure 3 is a flow chart diagram illustrating an example method for training a machine-learned model according to example implementations of aspects of the present disclosure;

[0014] Figure 4 is a block diagram of an example processing flow for using machine-learned model(s) to process input(s) to generate output(s) according to example implementations of aspects of the present disclosure;

[0015] Figure 5 is a block diagram of an example sequence processing model according to example implementations of aspects of the present disclosure;

[0016] Figure 6 is a block diagram of an example technique for populating an example input sequence for processing by a sequence processing model according to example implementations of aspects of the present disclosure;

[0017] Figure 7 is a block diagram of an example model development platform according to example implementations of aspects of the present disclosure;

[0018] Figure 8 is a block diagram of an example training workflow for training a machine-learned model according to example implementations of aspects of the present disclosure;

[0019] Figure 9 is a block diagram of an inference system for operating one or more machine-learned model(s) to perform inference according to example implementations of aspects of the present disclosure;

[0020] Figure 10 is a block diagram of an example networked computing system according to example implementations of aspects of the present disclosure;

[0021] Figure 11 is a block diagram of an example computing device according to example implementations of aspects of the present disclosure; and

[0022] Figure 12 is a block diagram of an example computing device according to example implementations of aspects of the present disclosure.

[0023] Reference numerals that are repeated across plural figures are intended to identify the same features in various implementations.

DETAILED DESCRIPTION

Overview

[0024] Example aspects of the present disclosure are directed to computer systems that enable machine-learned language models to choose the correct structural tools to leverage when solving challenging tasks. In particular, current language models rely heavily on internal knowledge to solve all downstream tasks, which causes hallucination and ungrounded answers. In contrast, the present disclosure proposes a new tool-using framework for language models, which enables the language models to smartly route a query to the most relevant pre-defined skills.

[0025] As one example, a computing system can generate a cache of embeddings associated with a set of previously-defined query-response pairs. For example, some or all of the previously-defined query-response pairs can demonstrate tool usage in service of generating an answer to the query. Then, when a new query is received, the computing system can generate a query embedding for the query and perform a similarity search to in the embedding space to identify and retrieve a number (e.g., the top- k) of the most similar previously-defined query-response pairs.

[0026] These retrieved previously-defined query-response pairs can be combined with the new query to generate a prompt for a machine-learned language model. The language model can process the prompt to generate a model response. For example, the model response can include one or more tokens that invoke the use of one or more structural tools such as computational tools (e.g., calculators), information retrieval tools (e.g., stored data, search engines, knowledge graphs), and/or programming tools (e.g., compilers, programming language interpreters, etc.). The computing system can process the model response (e.g., by executing the structural tools invoked by the model response) to generate a system response that is responsive to the query.

[0027] More particularly, access to and the correct use of different structural tools significantly increases the ability to solve challenging problems reliably and efficiently. Enabling language models to use tools is thus appealing in that: 1) the truthfulness of the language model completion is typically more trust-worthy, since execution by tools is

deterministic rather than purely probabilistic; 2) the size of language models can potentially be reduced since the domain-specific knowledge is delegated to external tools; and 3) tool-using language models are nearly immune to out-of-distribution performance degradation since by definition they are less dependent on the specific input that is provided.

[0028] As such, the present disclosure proposes a new general-purpose framework that enables language models (e.g., existing pre-trained language models) to use external tools efficiently. In some implementations, the proposed framework can first include the creation of a set of pre-defined query-response pairs. In some implementations, these query-response pairs can be referred to as “prototypes” and can correspond to or be derived from accumulated questions and answers during validation error analysis.

[0029] Next, to enable language models to perform a routing function for tool-using, a computing system implementing the proposed framework can embed each query in the set of prototypes with an embedding model (e.g., a pre-trained text embedding model). Then, when a new query is received, the query can similarly be embedded using the embedding model.

[0030] The computing system can generate a prompt used to trigger language model completion by concatenating one or more re-ranked prototypes whose questions are the top- k most similar ones with the newly received query. The language model can process the prompt to generate a model response. After language model completion, the computing system can, if appropriate, execute the generated model response with the support of a “toolbox” which contains the required tools (e.g., which may include data, libraries, and functions) for the successful execution of the model response.

[0031] Example experiments demonstrate that the proposed framework enables language models to greatly improve their ability to correctly respond to a more diverse and complex array of queries. In particular, example experiments demonstrate that the proposed framework outperforms current state-of-the-art methods on a number of different tasks that require complex processing.

[0032] The systems and methods of the present disclosure provide a number of technical effects and benefits. As one example technical effect and benefit, the proposed approach enables the language model to leverage structural tools to access additional information such as additional factual information. Thus, the language model can call and use such structural tools to have access to additional information which may be up-to-date, factual, domain-specific, client- or user-specific, etc. This improves the knowledge available to the language model when formulating the textual output and further improves the flexibility of the system by enabling the introduction of various information sources for various use cases.

[0033] In addition to improving the quality of the model's outputs, the proposed use of structural tools also leads to conservation of computational resources such as processor usage, memory usage, network bandwidth etc. Specifically, the knowledge available to previous language models was constrained to the information contained in (and learned from) their training datasets and/or information introduced within the contextual text input. Thus, their knowledge of factual information is severely limited and generally frozen in time. As such, when requested to produce an output that contains factual information, the models typically either hallucinate incorrect facts or supply outdated information. Therefore, an entire language model would need to be re-trained in order to keep language models up-to-date on changing real-world facts, to port the language model into a new domain or set of user information, or otherwise deploy a model in a new situation in which new information was at issue. Re-training of a language model requires the use of computational resources such as processor usage, memory usage, network bandwidth, etc.

[0034] However, the use of structural tools proposed by the present disclosure obviates the need to re-train the model in order to keep language models up-to-date on changing real-world facts, to port the language model into a new domain or set of user information, or otherwise deploy a model in a new situation in which new information was at issue. Instead, the model can simply be given access (e.g., via structural tools) to additional information which may be up-to-date, factual, domain-specific, client- or user-specific, etc. Thus, the model can easily be ported to different domains, uses, users, etc. and/or can provide responses which leverage up-to-date factual information without the need to re-train the model, thereby significantly conserving computational resources. By encoding the context in the form of intermediate analysis which may interface with (potentially external) information sources, the process may contribute to the resolution of technical constraints in the provision of information and/or functionality.

[0035] Similarly, another example technical effect is derived from the model's ability to leverage external sources to obtain information, rather than needing to store (e.g., in the form of learned relationships) all of the information needed to respond to various inputs. In particular, past approaches required storage and use (e.g., on a user device with constrained memory and/or battery availability) of a large model which had sufficient size (e.g., number of parameters) to learn and store relationships among various inputs and outputs. In contrast, some example implementations of the present disclosure can enable a "thin" (smaller) model to live on a user device or other mobile client or browser. The thin model can leverage various structural tools (e.g., cloud services) to save battery, compute, storage, updating, etc.

Thus, smaller models with access to structural tools can achieve similar or superior performance to large self-contained models, thereby saving computing resources such as memory usage, network bandwidth, energy consumption, etc. More generally,

[0036] As another example, the proposed techniques enable a machine learning model to better select a tool to use from a number of available tools. In particular, by providing useful examples showing correct tool usage, the model can automatically select the appropriate tool to use, out of multiple different tools that may be available. When the model is more confident it may call one tool instead of multiple different tools. This can conserve computing resources as the number of interactions between a model and different tools can be reduced.

[0037] As another example, the proposed models demonstrate improved interpretability. For example, the model response generated by the model can be reviewed or inspected (e.g., before or after execution) to interpret or understand how the final system response was generated in response to the contextual input. Improved interpretability can lead to more efficient use of computational resources such as processor usage, memory usage, etc. For example, a lack of interpretability in language model outputs can result in a lack of confidence or reliance on the model outputs, which can result in unnecessary overhead or other effort (e.g., computerized operations) which attempt to “double-check” the veracity or utility of the model’s output. By improving interpretability, confidence in computerized systems can be improved. In particular, reliability of the system responses may be verified and/or assessed to establish usability of the system for particular tasks.

[0038] With reference now to the Figures, example embodiments of the present disclosure will be discussed in further detail.

Example Model and Tool Usage

[0039] Aspects of the present disclosure are directed to a process in which in-context demonstration samples are used to prompt a language model to imitate the workflow presented in the few-shot samples, thereby enabling the language model to solve previously unseen tasks. In particular, some example implementations include three major stages: (a) pre-defining and embedding the prototypes for solving the task, (b) during inference, for each new query question, searching the most relevant k questions in the prototypes, and re-ranking them to inject into prompts, and (c) executing the generated content with the support of a pre-loaded toolbox to get the final results.

[0040] Example Accumulation of Prototypes from Errors

[0041] One aspect of the present disclosure is directed to the accumulation and usage of demonstrative query-response pairs, which may be referred to as “prototypes”. In some implementations, to find the prototypes that can lead to the best performance, for each of a number of evaluation tasks, a corresponding dataset can be split into test and validation sets by a certain ratio (e.g., 4:1). Then, additional prototypes can be accumulated by maintaining a “prototypes pool” containing the prototypes wrongly answered in every validation round.

[0042] One example procedure for accumulating prototypes from errors can be described as Algorithm 1:

[0043] **Algorithm 1 Accumulating Prototypes from Errors**

[0044] **Input:** $valid_data = \{x_i, y_i\}_{i=1\dots N}$, $valid_acc = 0$,
 $evaluating\ LM, review_manual = \{x_j, y_j\}_{j=1\dots m}$. Initialize $flag = false$.

[0045] **repeat**

[0046] **for** $i = 1$ **to** N **do**

[0047] Evaluate LM on $valid_data$ with $review_manual$.

[0048] **if** $LM(x_i) \neq y_i$ **then**

[0049] Write solution to x_i (which is y_i).

[0050] Append the new $\{x_i, y_i\}$ in $review_manual$.

[0051] **end if**

[0052] **end for**

[0053] Calculate $valid_acc$ on the $valid_data$.

[0054] **if** $valid_acc$ does not improve over k rounds **then**

[0055] $flag = true$.

[0056] **end if**

[0057] **until** $flag$ is $true$

[0058] In some implementations, a prototypes pool can be initialized with some randomly selected questions and answers. A stopping criteria can be established for the accumulating, which is whether the accuracy on the validation set does not increase after k rounds of validation with the updated prototypes pool. In one example, $k = 3$ as this “patience threshold”.

[0059] Example Routed Tool-Using Prompting

[0060] Given several prototypes from the prototypes pool (e.g., as conditioning inputs included in the prompt), a pre-trained language model (LM) can solve some of the questions

by imitating the workflow presented in the prototypes. Example experiments indicated that the model's success can greatly increase by actively selecting and inputting prototypes that have a similar workflow to the workflow that is needed to answer the query question. Therefore, one aspect of the present disclosure is directed to a routing system to smartly pick the proper prototypes for a given query question.

[0061] In particular, before the evaluation of a certain task starts, some example systems can first pre-embed all the questions in the stored prototypes with an embedding system, and save their embeddings in a cache file. Then, during evaluation, every new query question will be first embedded with the same embedding system. The computing system can then compute a distance metric (e.g., pairwise cosine similarity) with the embeddings of the prototype questions. This distance metric (e.g., cosine similarity) can be referred to as the relevance score. Finally, the computing system include some number (e.g., the top-*k*) of the most similar prototypes in the prompt used to trigger LM generation.

[0062] Example Execution for Verification

[0063] Once receiving the prompt prepared by the routing module, the LM can complete the generation of the query question by considering the workflow presented in the prototypes included in the prompt.

[0064] In some example implementations, most of the answers provided in the prototypes are code, since tool-using can be simply implemented by executing the code. To this end, some example implementations can include the configuration and use of a "toolbox" for each task. For example, the toolbox can correspond to a piece of global memory that can load data, code snippets, and/or random parameters to aid the execution of the LM-generated code.

[0065] As one example, the following items can be pre-loaded into the toolbox: a dataset file (e.g., a .csv file), a solver class which implements some desired functionality (e.g., the classic Dijkstra algorithm), and some packages needed for successful execution (e.g., pandas, numpy, etc.). These pre-loaded codes can be executed once before the task evaluation starts. For the generated code (e.g., the model response generated by the LM, referred to as *LM_gen*) for each question, the computing system can run *exec(LM_gen, globals())* to get verified results.

[0066] In some implementations, the computing system use *eval('answer')* to extract the value stored in the answer variable to get the final answer (e.g., the LM is expected to

imitate the answer in the prototypes that also stores the answer in the variable *answer* in the final step).

[0067] Example Query Response System

[0068] Figure 1 depicts an example query response system 110 that implements the concepts described above. In particular, Figure 1 depicts a block diagram of an example data flow to generate a system response 136 to a query 112 according to example embodiments of the present disclosure.

[0069] As shown in Figure 1, a computing system can obtain data descriptive of a query 112. For example, the query 112 can be received from a user. For example, the query can be a natural language question.

[0070] The computing system can generate a query embedding 116 for the query 112 using an embedding model 114. For example, the query embedding 116 can be expressed within a latent embedding space.

[0071] The computing system can perform a similarity search 120 for the query embedding 116 within the latent embedding space to identify one or more previously-defined embeddings 124 associated with one or more previously-defined query-response pairs 118. For example, performing the similarity search 120 can include ranking the one or more previously-defined query-response pairs 118. For example, the previously-defined query-response pairs 118 can be ranked based on a distance measure (e.g., pairwise cosine similarity) evaluated between the embedding 122 for each pair 118 and the query embedding 116. In some implementations, performing the similarity search 120 can include identifying a top-*k* set of the previously-defined embeddings 122 from the embedding space, where *k* is a hyperparameter. In other implementations, all embeddings 122 that have a distance measure less than some threshold can be returned.

[0072] More particularly, the computing system can generate embeddings 122 for the previously-defined query-response pairs 118 using the embedding model 114. For example, this process can be done offline prior to receipt of the query 112 and the embeddings 122 can be stored in a database for use when the query 112 is received. For example, in some implementations, the embeddings 122 can be generated from only the query portion of the previously-defined query-response pairs 118. According to an aspect of the present disclosure, some or all of the previously-defined query-response pairs 118 can demonstrate usage of one or more structural tools 138.

[0073] In some implementations, certain subsets of the previously-defined query-response pairs 18 can be associated with different structural tools 138. For example, a first

pair may be an example of usage of a first tool while a second pair may be an example of usage of a second tool. A user associated with the query 112 may have access to or otherwise be permitted to use some but not all of the structural tools 138. In some implementations, the similarity search 120 can be limited to searching against only previously-defined embeddings 124 generated from query-response pairs 118 that are associated with structural tools 138 for which that the user has access or permission to use. In addition or alternatively, a user may be enabled to select (e.g., via a user interface) one or more of the structural tools 138. In some implementations, the similarity search 120 can be limited to searching against only previously-defined embeddings 124 generated from query-response pairs 18 that are associated with the one or more of the structural tools 138 selected by the user.

[0074] The computing system can generate (e.g., by performing prompt construction 126) a prompt 128 based on the query 112 and the one or more previously-defined query-response pairs 124 identified by the similarity search 120. As one example, the computing system can concatenate the query 112 with the one or more previously-defined query-response pairs 124 identified by the similarity search 120.

[0075] The computing system can provide the prompt 128 as an input for processing by a machine-learned language model 130. For example, the language model 130 can be a pre-trained large language model such as, as examples, the BERT model, the LaMDA model, the PaLM model, etc. The computing system can receive a model response 132 to the query 112 that was output by the machine-learned language model 130 based on processing of the prompt 128.

[0076] The computing system can perform response execution 134 to process the model response 132 to generate the system response 136. The system response 136 can be provided as an output to the user. For example, the system response 136 can be a natural language answer to a natural language question.

[0077] According to an aspect of the present disclosure, in some implementations, the model response 132 can include one or more tokens that, when executed by the computing system (e.g., shown as response execution 134), cause the structural tool 138 to retrieve or transform information. Thus, in some implementations, performing the response execution 134 to process the model response 132 can include executing the one or more tokens to cause the structural tool 138 to retrieve or transform the information. The system response 136 can be based at least in part on the information retrieved or transformed by the structural tool 138 responsive to execution of the one or more tokens.

[0078] In some implementations, a toolbox 140 can be provided for use by the structural tool(s) 138. For example, the toolbox 140 can include any knowledge, data, or information that enables the structural tool to retrieve or transform information as requested or instructed by the model response 132.

[0079] To provide an illustrative example, in some implementations, the model response 132 can include instructions expressed in a computer language (e.g., an executable computer program) and the structural tool 138 can be a programming language interpreter configured to execute the instructions expressed in the computer language. In some of such implementations, the toolbox 140 can include pre-loaded information such as one or more libraries or datasets associated with the computer language. The one or more libraries or datasets can be provided in a shared computer environment with the programming language interpreter. As one example, the computer language can be a Python computer language.

[0080] As other examples, the structural tool 138 can be, include, or leverage: a database lookup to access additional information from a database; an application programming interface (API) call to request and receive additional information via the API; a query service that queries results from a search engine, knowledge graph, or digital assistant; a calculator tool; or other computational tools.

[0081] In some implementations, the machine-learned language model 130 can optionally be fine-tuned on the previously-defined query-response pairs 118 prior to using the model 130 as illustrated in Figure 1.

[0082] Example Illustration

[0083] Figure 2 depicts a graphical diagram of a specific example process to generate a system response to a query according to example embodiments of the present disclosure. As shown in portion (a), a computing system can first accumulate a set of question-answer pairs (named “prototype pool”) from validation errors. As shown in portion (b), the computing system can embed the questions in the prototype pool, and every time the LM is queried with a new question, the computing system search the most similar k questions in the review manual, rerank the corresponding prototypes, and concatenate them as the prompt injection. As shown in portion (c), the LM generation triggered by the prompt can be executed (e.g., by a Python interpreter) in an environment that pre-loads data, libraries, code snippets, etc. that can support a certain task. Finally, the answer can be collected from the variables in the runtime.

[0084] More particularly, Figure 2 illustrates a graphical diagram of an example process to generate a system response to a query according to example embodiments of the present

disclosure. The process depicted in Figure 2 exemplifies the interaction between pre-defined query-response pairs, known as prototypes, and a new user query, which leads to the generation of a system response through the use of a machine-learned language model and a toolbox of resources.

[0085] In the first stage (a), the process involves the selection of relevant prototypes (e.g., Prototype 1, Prototype 4, and Prototype 8) based on their relevance scores, which indicate the degree of similarity to the new user query. Each prototype contains a question and an answer, where the answer can be in the form of non-executable natural language and/or in the form of executable programs. For example, Prototype 4 with a relevance score of 0.6 provides an answer in executable code to determine the number of flights operated by Delta, while Prototype 1 with a higher relevance score of 0.9 demonstrates the use of a DijkstraSolver class to find the cheapest flight under certain constraints.

[0086] The second stage (b) involves the combination of the new user query with the reranked prototypes to form a prompt that is injected into the language model. The example user query expresses a desire to find the lowest price for a flight from SFO to JFK with preferences for Delta flights and a maximum of two stops. The reranked prototypes are selected to aid the language model in understanding the context and requirements of the query by providing examples of similar problems and their solutions.

[0087] In the final stage (c), the language model processes the combined prompt of the new query and selected prototypes to compose a model response. The response leverages both composition and planning skills demonstrated in the prototypes, such as filtering a dataset for Delta flights and using a planning algorithm to find the cheapest flight option within given constraints. The model response, which includes executable code, is then run using the toolbox for flights booking. The toolbox contains essential resources such as the flights dataset, internal code snippets like the DijkstraSolver class, and external libraries, for instance, pandas. The execution results in a system response with the lowest flight price, confirming the successful application of the method and tools to solve the user's query.

[0088] This interactive execution process serves as an example embodiment of the proposed techniques, showcasing how a computing system can efficiently use structural tools to generate accurate and up-to-date responses to complex queries by leveraging machine-learned language models and pre-defined query-response prototypes.

Example Methods

[0089] Figure 3 depicts a flowchart of a method 300 for training one or more machine-learned models according to aspects of the present disclosure. For instance, an example machine-learned model can include a {reference to claimed model(s)}

[0090] One or more portion(s) of example method 300 can be implemented by a computing system that includes one or more computing devices such as, for example, computing systems described with reference to the other figures. Each respective portion of example method 300 can be performed by any (or any combination) of one or more computing devices. Moreover, one or more portion(s) of example method 300 can be implemented on the hardware components of the device(s) described herein, for example, to train one or more systems or models. Figure 3 depicts elements performed in a particular order for purposes of illustration and discussion. Those of ordinary skill in the art, using the disclosures provided herein, will understand that the elements of any of the methods discussed herein can be adapted, rearranged, expanded, omitted, combined, or modified in various ways without deviating from the scope of the present disclosure. Figure 3 is described with reference to elements/terms described with respect to other systems and figures for exemplary illustrated purposes and is not meant to be limiting. One or more portions of example method 300 can be performed additionally, or alternatively, by other systems.

[0091] At 302, example method 300 can include obtaining a training instance. A set of training data can include a plurality of training instances divided between multiple datasets (e.g., a training dataset, a validation dataset, or testing dataset). A training instance can be labeled or unlabeled. Although referred to in example method 300 as a “training” instance, it is to be understood that runtime inferences can form training instances when a model is trained using an evaluation of the model’s performance on that runtime instance (e.g., online training/learning). Example data types for the training instance and various tasks associated therewith are described throughout the present disclosure.

[0092] At 304, example method 300 can include processing, using one or more machine-learned models, the training instance to generate an output. The output can be directly obtained from the one or more machine-learned models or can be a downstream result of a chain of processing operations that includes an output of the one or more machine-learned models.

[0093] At 306, example method 300 can include receiving an evaluation signal associated with the output. The evaluation signal can be obtained using a loss function. Various determinations of loss can be used, such as mean squared error, likelihood loss, cross

entropy loss, hinge loss, contrastive loss, or various other loss functions. The evaluation signal can be computed using known ground-truth labels (e.g., supervised learning), predicted or estimated labels (e.g., semi- or self-supervised learning), or without labels (e.g., unsupervised learning). The evaluation signal can be a reward (e.g., for reinforcement learning). The reward can be computed using a machine-learned reward model configured to generate rewards based on output(s) received. The reward can be computed using feedback data describing human feedback on the output(s).

[0094] At 308, example method 300 can include updating the machine-learned model using the evaluation signal. For example, values for parameters of the machine-learned model(s) can be learned, in some embodiments, using various training or learning techniques, such as, for example, backwards propagation. For example, the evaluation signal can be backpropagated from the output (or another source of the evaluation signal) through the machine-learned model(s) to update one or more parameters of the model(s) (e.g., based on a gradient of the evaluation signal with respect to the parameter value(s)). For example, system(s) containing one or more machine-learned models can be trained in an end-to-end manner. Gradient descent techniques can be used to iteratively update the parameters over a number of training iterations. In some implementations, performing backwards propagation of errors can include performing truncated backpropagation through time. Example method 300 can include implementing a number of generalization techniques (e.g., weight decays, dropouts, etc.) to improve the generalization capability of the models being trained.

[0095] In some implementations, example method 300 can be implemented for training a machine-learned model from an initialized state to a fully trained state (e.g., when the model exhibits a desired performance profile, such as based on accuracy, precision, recall, etc.).

[0096] In some implementations, example method 300 can be implemented for particular stages of a training procedure. For instance, in some implementations, example method 300 can be implemented for pre-training a machine-learned model. Pre-training can include, for instance, large-scale training over potentially noisy data to achieve a broad base of performance levels across a variety of tasks/data types. In some implementations, example method 300 can be implemented for fine-tuning a machine-learned model. Fine-tuning can include, for instance, smaller-scale training on higher-quality (e.g., labeled, curated, etc.) data. Fine-tuning can affect all or a portion of the parameters of a machine-learned model. For example, various portions of the machine-learned model can be “frozen” for certain training stages. For example, parameters associated with an embedding space can be “frozen” during fine-tuning (e.g., to retain information learned from a broader domain(s) than present

in the fine-tuning dataset(s)). An example fine-tuning approach includes reinforcement learning. Reinforcement learning can be based on user feedback on model performance during use.

Example Machine-Learned Models

[0097] Figure 4 is a block diagram of an example processing flow for using machine-learned model(s) 1 to process input(s) 2 to generate output(s) 3.

[0098] Machine-learned model(s) 1 can be or include one or multiple machine-learned models or model components. Example machine-learned models can include neural networks (e.g., deep neural networks). Example machine-learned models can include non-linear models or linear models. Example machine-learned models can use other architectures in lieu of or in addition to neural networks. Example machine-learned models can include decision tree based models, support vector machines, hidden Markov models, Bayesian networks, linear regression models, k-means clustering models, etc.

[0099] Example neural networks can include feed-forward neural networks, recurrent neural networks (RNNs), including long short-term memory (LSTM) based recurrent neural networks, convolutional neural networks (CNNs), diffusion models, generative-adversarial networks, or other forms of neural networks. Example neural networks can be deep neural networks. Some example machine-learned models can leverage an attention mechanism such as self-attention. For example, some example machine-learned models can include multi-headed self-attention models.

[0100] Machine-learned model(s) 1 can include a single or multiple instances of the same model configured to operate on data from input(s) 2. Machine-learned model(s) 1 can include an ensemble of different models that can cooperatively interact to process data from input(s) 2. For example, machine-learned model(s) 1 can employ a mixture-of-experts structure. See, e.g., Zhou et al., Mixture-of-Experts with Expert Choice Routing, arXiv:2202.09368v2 (Oct. 14, 2022).

[0101] Input(s) 2 can generally include or otherwise represent various types of data. Input(s) 2 can include one type or many different types of data. Output(s) 3 can be data of the same type(s) or of different types of data as compared to input(s) 2. Output(s) 3 can include one type or many different types of data.

[0102] Example data types for input(s) 2 or output(s) 3 include natural language text data, software code data (e.g., source code, object code, machine code, or any other form of computer-readable instructions or programming languages), machine code data (e.g., binary

code, assembly code, or other forms of machine-readable instructions that can be executed directly by a computer's central processing unit), assembly code data (e.g., low-level programming languages that use symbolic representations of machine code instructions to program a processing unit), genetic data or other chemical or biochemical data, image data, audio data, audiovisual data, haptic data, biometric data, medical data, financial data, statistical data, geographical data, astronomical data, historical data, sensor data generally (e.g., digital or analog values, such as voltage or other absolute or relative level measurement values from a real or artificial input, such as from an audio sensor, light sensor, displacement sensor, etc.), and the like. Data can be raw or processed and can be in any format or schema.

[0103] In multimodal inputs 2 or outputs 3, example combinations of data types include image data and audio data, image data and natural language data, natural language data and software code data, image data and biometric data, sensor data and medical data, etc. It is to be understood that any combination of data types in an input 2 or an output 3 can be present.

[0104] An example input 2 can include one or multiple data types, such as the example data types noted above. An example output 3 can include one or multiple data types, such as the example data types noted above. The data type(s) of input 2 can be the same as or different from the data type(s) of output 3. It is to be understood that the example data types noted above are provided for illustrative purposes only. Data types contemplated within the scope of the present disclosure are not limited to those examples noted above.

Example Machine-Learned Sequence Processing Models

[0105] Figure 5 is a block diagram of an example implementation of an example machine-learned model configured to process sequences of information. For instance, an example implementation of machine-learned model(s) 1 can include machine-learned sequence processing model(s) 4. An example system can pass input(s) 2 to sequence processing model(s) 4. Sequence processing model(s) 4 can include one or more machine-learned components. Sequence processing model(s) 4 can process the data from input(s) 2 to obtain an input sequence 5. Input sequence 5 can include one or more input elements 5-1, 5-2, . . . , 5-M, etc. obtained from input(s) 2. Sequence processing model 4 can process input sequence 5 using prediction layer(s) 6 to generate an output sequence 7. Output sequence 7 can include one or more output elements 7-1, 7-2, . . . , 7-N, etc. generated based on input sequence 5. The system can generate output(s) 3 based on output sequence 7.

[0106] Sequence processing model(s) 4 can include one or multiple machine-learned model components configured to ingest, generate, or otherwise reason over sequences of

information. For example, some example sequence processing models in the text domain are referred to as “Large Language Models,” or LLMs. See, e.g., PaLM 2 Technical Report, Google, <https://ai.google/static/documents/palm2techreport.pdf> (n.d.). Other example sequence processing models can operate in other domains, such as image domains, see, e.g., Dosovitskiy et al., An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale, arXiv:2010.11929v2 (Jun. 3, 2021), audio domains, see, e.g., Agostinelli et al., MusicLM: Generating Music From Text, arXiv:2301.11325v1 (Jan. 26, 2023), biochemical domains, see, e.g., Jumper et al., Highly accurate protein structure prediction with AlphaFold, 596 Nature 583 (Aug. 26, 2021), by way of example. Sequence processing model(s) 4 can process one or multiple types of data simultaneously. Sequence processing model(s) 4 can include relatively large models (e.g., more parameters, computationally expensive, etc.), relatively small models (e.g., fewer parameters, computationally lightweight, etc.), or both.

[0107] In general, sequence processing model(s) 4 can obtain input sequence 5 using data from input(s) 2. For instance, input sequence 5 can include a representation of data from input(s) 2 in a format understood by sequence processing model(s) 4. One or more machine-learned components of sequence processing model(s) 4 can ingest the data from input(s) 2, parse the data into pieces compatible with the processing architectures of sequence processing model(s) 4 (e.g., via “tokenization”), and project the pieces into an input space associated with prediction layer(s) 6 (e.g., via “embedding”).

[0108] Sequence processing model(s) 4 can ingest the data from input(s) 2 and parse the data into a sequence of elements to obtain input sequence 5. For example, a portion of input data from input(s) 2 can be broken down into pieces that collectively represent the content of the portion of the input data. The pieces can provide the elements of the sequence.

[0109] Elements 5-1, 5-2, . . . , 5-M can represent, in some cases, building blocks for capturing or expressing meaningful information in a particular data domain. For instance, the elements can describe “atomic units” across one or more domains. For example, for textual input source(s), the elements can correspond to groups of one or more words or sub-word components, such as sets of one or more characters.

[0110] For example, elements 5-1, 5-2, . . . , 5-M can represent tokens obtained using a tokenizer. For instance, a tokenizer can process a given portion of an input source and output a series of tokens (e.g., corresponding to input elements 5-1, 5-2, . . . , 5-M) that represent the portion of the input source. Various approaches to tokenization can be used. For instance, textual input source(s) can be tokenized using a byte-pair encoding (BPE) technique. See, e.g., Kudo et al., SentencePiece: A simple and language independent subword tokenizer and

detokenizer for Neural Text Processing, Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing (System Demonstrations), pages 66–71 (October 31–November 4, 2018), <https://aclanthology.org/D18-2012.pdf>. Image-based input source(s) can be tokenized by extracting and serializing patches from an image.

[0111] In general, arbitrary data types can be serialized and processed into input sequence 5. It is to be understood that element(s) 5-1, 5-2, . . . , 5-M depicted in Figure 5 can be the tokens or can be the embedded representations thereof.

[0112] Prediction layer(s) 6 can predict one or more output elements 7-1, 7-2, . . . , 7-N based on the input elements. Prediction layer(s) 6 can include one or more machine-learned model architectures, such as one or more layers of learned parameters that manipulate and transform the input(s) to extract higher-order meaning from, and relationships between, input element(s) 5-1, 5-2, . . . , 5-M. In this manner, for instance, example prediction layer(s) 6 can predict new output element(s) in view of the context provided by input sequence 5.

[0113] Prediction layer(s) 6 can evaluate associations between portions of input sequence 5 and a particular output element. These associations can inform a prediction of the likelihood that a particular output follows the input context. For example, consider the textual snippet, “The carpenter’s toolbox was small and heavy. It was full of ____.” Example prediction layer(s) 6 can identify that “It” refers back to “toolbox” by determining a relationship between the respective embeddings. Example prediction layer(s) 6 can also link “It” to the attributes of the toolbox, such as “small” and “heavy.” Based on these associations, prediction layer(s) 6 can, for instance, assign a higher probability to the word “nails” than to the word “sawdust.”

[0114] A transformer is an example architecture that can be used in prediction layer(s) 4. See, e.g., Vaswani et al., Attention Is All You Need, arXiv:1706.03762v7 (Aug. 2, 2023). A transformer is an example of a machine-learned model architecture that uses an attention mechanism to compute associations between items within a context window. The context window can include a sequence that contains input sequence 5 and potentially one or more output element(s) 7-1, 7-2, . . . , 7-N. A transformer block can include one or more attention layer(s) and one or more post-attention layer(s) (e.g., feedforward layer(s), such as a multi-layer perceptron).

[0115] Prediction layer(s) 6 can include other machine-learned model architectures in addition to or in lieu of transformer-based architectures. For example, recurrent neural networks (RNNs) and long short-term memory (LSTM) models can also be used, as well as

convolutional neural networks (CNNs). In general, prediction layer(s) 6 can leverage various kinds of artificial neural networks that can understand or generate sequences of information.

[0116] Output sequence 7 can include or otherwise represent the same or different data types as input sequence 5. For instance, input sequence 5 can represent textual data, and output sequence 7 can represent textual data. Input sequence 5 can represent image, audio, or audiovisual data, and output sequence 7 can represent textual data (e.g., describing the image, audio, or audiovisual data). It is to be understood that prediction layer(s) 6, and any other interstitial model components of sequence processing model(s) 4, can be configured to receive a variety of data types in input sequence(s) 5 and output a variety of data types in output sequence(s) 7.

[0117] Output sequence 7 can have various relationships to input sequence 5. Output sequence 7 can be a continuation of input sequence 5. Output sequence 7 can be complementary to input sequence 5. Output sequence 7 can translate, transform, augment, or otherwise modify input sequence 5. Output sequence 7 can answer, evaluate, confirm, or otherwise respond to input sequence 5. Output sequence 7 can implement (or describe instructions for implementing) an instruction provided via input sequence 5.

[0118] Output sequence 7 can be generated autoregressively. For instance, for some applications, an output of one or more prediction layer(s) 6 can be passed through one or more output layers (e.g., softmax layer) to obtain a probability distribution over an output vocabulary (e.g., a textual or symbolic vocabulary) conditioned on a set of input elements in a context window. In this manner, for instance, output sequence 7 can be autoregressively generated by sampling a likely next output element, adding that element to the context window, and re-generating the probability distribution based on the updated context window, and sampling a likely next output element, and so forth.

[0119] Output sequence 7 can also be generated non-autoregressively. For instance, multiple output elements of output sequence 7 can be predicted together without explicit sequential conditioning on each other. See, e.g., Saharia et al., Non-Autoregressive Machine Translation with Latent Alignments, arXiv:2004.07437v3 (Nov. 16, 2020).

[0120] Output sequence 7 can include one or multiple portions or elements. In an example content generation configuration, output sequence 7 can include multiple elements corresponding to multiple portions of a generated output sequence (e.g., a textual sentence, values of a discretized waveform, computer code, etc.). In an example classification configuration, output sequence 7 can include a single element associated with a classification output. For instance, an output “vocabulary” can include a set of classes into which an input

sequence is to be classified. For instance, a vision transformer block can pass latent state information to a multilayer perceptron that outputs a likely class value associated with an input image.

[0121] Figure 6 is a block diagram of an example technique for populating an example input sequence 8. Input sequence 8 can include various functional elements that form part of the model infrastructure, such as an element 8-0 obtained from a task indicator 9 that signals to any model(s) that process input sequence 8 that a particular task is being performed (e.g., to help adapt a performance of the model(s) to that particular task). Input sequence 8 can include various data elements from different data modalities. For instance, an input modality 10-1 can include one modality of data. A data-to-sequence model 11-1 can process data from input modality 10-1 to project the data into a format compatible with input sequence 8 (e.g., one or more vectors dimensioned according to the dimensions of input sequence 8) to obtain elements 8-1, 8-2, 8-3. Another input modality 10-2 can include a different modality of data. A data-to-sequence model 11-2 can project data from input modality 10-2 into a format compatible with input sequence 8 to obtain elements 8-4, 8-5, 8-6. Another input modality 10-3 can include yet another different modality of data. A data-to-sequence model 11-3 can project data from input modality 10-3 into a format compatible with input sequence 8 to obtain elements 8-7, 8-8, 8-9.

[0122] Input sequence 8 can be the same as or different from input sequence 5. Input sequence 8 can be a multimodal input sequence that contains elements that represent data from different modalities using a common dimensional representation. For instance, an embedding space can have P dimensions. Input sequence 8 can be configured to contain a plurality of elements that have P dimensions. In this manner, for instance, example implementations can facilitate information extraction and reasoning across diverse data modalities by projecting data into elements in the same embedding space for comparison, combination, or other computations therebetween.

[0123] For example, elements 8-0, . . . , 8-9 can indicate particular locations within a multidimensional embedding space. Some elements can map to a set of discrete locations in the embedding space. For instance, elements that correspond to discrete members of a predetermined vocabulary of tokens can map to discrete locations in the embedding space that are associated with those tokens. Other elements can be continuously distributed across the embedding space. For instance, some data types can be broken down into continuously defined portions (e.g., image patches) that can be described using continuously distributed locations within the embedding space.

[0124] In some implementations, the expressive power of the embedding space may not be limited to meanings associated with any particular set of tokens or other building blocks. For example, a continuous embedding space can encode a spectrum of high-order information. An individual piece of information (e.g., a token) can map to a particular point in that space: for instance, a token for the word “dog” can be projected to an embedded value that points to a particular location in the embedding space associated with canine-related information. Similarly, an image patch of an image of a dog on grass can also be projected into the embedding space. In some implementations, the projection of the image of the dog can be similar to the projection of the word “dog” while also having similarity to a projection of the word “grass,” while potentially being different from both. In some implementations, the projection of the image patch may not exactly align with any single projection of a single word. In some implementations, the projection of the image patch can align with a combination of the projections of the words “dog” and “grass.” In this manner, for instance, a high-order embedding space can encode information that can be independent of data modalities in which the information is expressed.

[0125] Task indicator 9 can include a model or model component configured to identify a task being performed and inject, into input sequence 8, an input value represented by element 8-0 that signals which task is being performed. For instance, the input value can be provided as a data type associated with an input modality and projected along with that input modality (e.g., the input value can be a textual task label that is embedded along with other textual data in the input; the input value can be a pixel-based representation of a task that is embedded along with other image data in the input; etc.). The input value can be provided as a data type that differs from or is at least independent from other input(s). For instance, the input value represented by element 8-0 can be a learned within a continuous embedding space.

[0126] Input modalities 10-1, 10-2, and 10-3 can be associated with various different data types (e.g., as described above with respect to input(s) 2 and output(s) 3).

[0127] Data-to-sequence models 11-1, 11-2, and 11-3 can be the same or different from each other. Data-to-sequence models 11-1, 11-2, and 11-3 can be adapted to each respective input modality 10-1, 10-2, and 10-3. For example, a textual data-to-sequence model can subdivide a portion of input text and project the subdivisions into element(s) in input sequence 8 (e.g., elements 8-1, 8-2, 8-3, etc.). An image data-to-sequence model can subdivide an input image and project the subdivisions into element(s) in input sequence 8 (e.g., elements 8-4, 8-5, 8-6, etc.). An arbitrary datatype data-to-sequence model can

subdivide an input of that arbitrary datatype and project the subdivisions into element(s) in input sequence 8 (e.g., elements 8-7, 8-8, 8-9, etc.).

[0128] Data-to-sequence models 11-1, 11-2, and 11-3 can form part of machine-learned sequence processing model(s) 4. Data-to-sequence models 11-1, 11-2, and 11-3 can be jointly trained with or trained independently from machine-learned sequence processing model(s) 4. Data-to-sequence models 11-1, 11-2, and 11-3 can be trained end-to-end with machine-learned sequence processing model(s) 4.

Example Machine-Learned Model Development Platform

[0129] Figure 7 is a block diagram of an example model development platform 12 that can facilitate creation, adaptation, and refinement of example machine-learned models (e.g., machine-learned model(s) 1, sequence processing model(s) 4, etc.). Model development platform 12 can provide a number of different toolkits that developer systems can employ in the development of new or adapted machine-learned models.

[0130] Model development platform 12 can provide one or more model libraries 13 containing building blocks for new models. Model libraries 13 can include one or more pre-trained foundational models 13-1, which can provide a backbone of processing power across various tasks. Model libraries 13 can include one or more pre-trained expert models 13-2, which can be focused on performance in particular domains of expertise. Model libraries 13 can include various model primitives 13-3, which can provide low-level architectures or components (optionally pre-trained), which can be assembled in various arrangements as desired.

[0131] Model development platform 12 can receive selections of various model components 14. Model development platform 12 can pass selected model components 14 to a workbench 15 that combines selected model components 14 into a development model 16.

[0132] Workbench 15 can facilitate further refinement and adaptation of development model 16 by leveraging a number of different toolkits integrated with model development platform 12. For example, workbench 15 can facilitate alignment of the development model 16 with a desired performance profile on various tasks using a model alignment toolkit 17.

[0133] Model alignment toolkit 17 can provide a number of tools for causing development model 16 to generate outputs aligned with desired behavioral characteristics. Alignment can include increasing an accuracy, precision, recall, etc. of model outputs. Alignment can include enforcing output styles, schema, or other preferential characteristics of model outputs. Alignment can be general or domain-specific. For instance, a pre-trained

foundational model 13-1 can begin with an initial level of performance across multiple domains. Alignment of the pre-trained foundational model 13-1 can include improving a performance in a particular domain of information or tasks (e.g., even at the expense of performance in another domain of information or tasks).

[0134] Model alignment toolkit 17 can integrate one or more dataset(s) 17-1 for aligning development model 16. Curated dataset(s) 17-1 can include labeled or unlabeled training data. Dataset(s) 17-1 can be obtained from public domain datasets. Dataset(s) 17-1 can be obtained from private datasets associated with one or more developer system(s) for the alignment of bespoke machine-learned model(s) customized for private use-cases.

[0135] Pre-training pipelines 17-2 can include a machine-learned model training workflow configured to update development model 16 over large-scale, potentially noisy datasets. For example, pre-training can leverage unsupervised learning techniques (e.g., de-noising, etc.) to process large numbers of training instances to update model parameters from an initialized state and achieve a desired baseline performance. Pre-training pipelines 17-2 can leverage unlabeled datasets in dataset(s) 17-1 to perform pre-training. Workbench 15 can implement a pre-training pipeline 17-2 to pre-train development model 16.

[0136] Fine-tuning pipelines 17-3 can include a machine-learned model training workflow configured to refine the model parameters of development model 16 with higher-quality data. Fine-tuning pipelines 17-3 can update development model 16 by conducting supervised training with labeled dataset(s) in dataset(s) 17-1. Fine-tuning pipelines 17-3 can update development model 16 by conducting reinforcement learning using reward signals from user feedback signals. Workbench 15 can implement a fine-tuning pipeline 17-3 to fine-tune development model 16.

[0137] Prompt libraries 17-4 can include sets of inputs configured to induce behavior aligned with desired performance criteria. Prompt libraries 17-4 can include few-shot prompts (e.g., inputs providing examples of desired model outputs for prepending to a desired runtime query), chain-of-thought prompts (e.g., inputs providing step-by-step reasoning within the exemplars to facilitate thorough reasoning by the model), and the like.

[0138] Example prompts can be retrieved from an available repository of prompt libraries 17-4. Example prompts can be contributed by one or more developer systems using workbench 15.

[0139] In some implementations, pre-trained or fine-tuned models can achieve satisfactory performance without exemplars in the inputs. For instance, zero-shot prompts can

include inputs that lack exemplars. Zero-shot prompts can be within a domain within a training dataset or outside of the training domain(s).

[0140] Prompt libraries 17-4 can include one or more prompt engineering tools. Prompt engineering tools can provide workflows for retrieving or learning optimized prompt values. Prompt engineering tools can facilitate directly learning prompt values (e.g., input element values) based one or more training iterations. Workbench 15 can implement prompt engineering tools in development model 16.

[0141] Prompt libraries 17-4 can include pipelines for prompt generation. For example, inputs can be generated using development model 16 itself or other machine-learned models. In this manner, for instance, a first model can process information about a task and output a input for a second model to process in order to perform a step of the task. The second model can be the same as or different from the first model. Workbench 15 can implement prompt generation pipelines in development model 16.

[0142] Prompt libraries 17-4 can include pipelines for context injection. For instance, a performance of development model 16 on a particular task can improve if provided with additional context for performing the task. Prompt libraries 17-4 can include software components configured to identify desired context, retrieve the context from an external source (e.g., a database, a sensor, etc.), and add the context to the input prompt. Workbench 15 can implement context injection pipelines in development model 16.

[0143] Although various training examples described herein with respect to model development platform 12 refer to “pre-training” and “fine-tuning,” it is to be understood that model alignment toolkit 17 can generally support a wide variety of training techniques adapted for training a wide variety of machine-learned models. Example training techniques can correspond to the example training method 300 described above.

[0144] Model development platform 12 can include a model plugin toolkit 18. Model plugin toolkit 18 can include a variety of tools configured for augmenting the functionality of a machine-learned model by integrating the machine-learned model with other systems, devices, and software components. For instance, a machine-learned model can use tools to increase performance quality where appropriate. For instance, deterministic tasks can be offloaded to dedicated tools in lieu of probabilistically performing the task with an increased risk of error. For instance, instead of autoregressively predicting the solution to a system of equations, a machine-learned model can recognize a tool to call for obtaining the solution and pass the system of equations to the appropriate tool. The tool can be a traditional system of equations solver that can operate deterministically to resolve the system of equations. The

output of the tool can be returned in response to the original query. In this manner, tool use can allow some example models to focus on the strengths of machine-learned models—e.g., understanding an intent in an unstructured request for a task—while augmenting the performance of the model by offloading certain tasks to a more focused tool for rote application of deterministic algorithms to a well-defined problem.

[0145] Model plugin toolkit 18 can include validation tools 18-1. Validation tools 18-1 can include tools that can parse and confirm output(s) of a machine-learned model.

Validation tools 18-1 can include engineered heuristics that establish certain thresholds applied to model outputs. For example, validation tools 18-1 can ground the outputs of machine-learned models to structured data sources (e.g., to mitigate “hallucinations”).

[0146] Model plugin toolkit 18 can include tooling packages 18-2 for implementing one or more tools that can include scripts or other executable code that can be executed alongside development model 16. Tooling packages 18-2 can include one or more inputs configured to cause machine-learned model(s) to implement the tools (e.g., few-shot prompts that induce a model to output tool calls in the proper syntax, etc.). Tooling packages 18-2 can include, for instance, fine-tuning training data for training a model to use a tool.

[0147] Model plugin toolkit 18 can include interfaces for calling external application programming interfaces (APIs) 18-3. For instance, in addition to or in lieu of implementing tool calls or tool code directly with development model 16, development model 16 can be aligned to output instruction that initiate API calls to send or obtain data via external systems.

[0148] Model plugin toolkit 18 can integrate with prompt libraries 17-4 to build a catalog of available tools for use with development model 16. For instance, a model can receive, in an input, a catalog of available tools, and the model can generate an output that selects a tool from the available tools and initiates a tool call for using the tool.

[0149] Model development platform 12 can include a computational optimization toolkit 19 for optimizing a computational performance of development model 16. For instance, tools for model compression 19-1 can allow development model 16 to be reduced in size while maintaining a desired level of performance. For instance, model compression 19-1 can include quantization workflows, weight pruning and sparsification techniques, etc. Tools for hardware acceleration 19-2 can facilitate the configuration of the model storage and execution formats to operate optimally on different hardware resources. For instance, hardware acceleration 19-2 can include tools for optimally sharding models for distributed processing over multiple processing units for increased bandwidth, lower unified memory requirements, etc. Tools for distillation 19-3 can provide for the training of lighter-weight

models based on the knowledge encoded in development model 16. For instance, development model 16 can be a highly performant, large machine-learned model optimized using model development platform 12. To obtain a lightweight model for running in resource-constrained environments, a smaller model can be a “student model” that learns to imitate development model 16 as a “teacher model.” In this manner, for instance, the investment in learning the parameters and configurations of development model 16 can be efficiently transferred to a smaller model for more efficient inference.

[0150] Workbench 15 can implement one, multiple, or none of the toolkits implemented in model development platform 12. Workbench 15 can output an output model 20 based on development model 16. Output model 20 can be a deployment version of development model 16. Output model 20 can be a development or training checkpoint of development model 16. Output model 20 can be a distilled, compressed, or otherwise optimized version of development model 16.

[0151] Figure 8 is a block diagram of an example training flow for training a machine-learned development model 16. One or more portion(s) of the example training flow can be implemented by a computing system that includes one or more computing devices such as, for example, computing systems described with reference to the other figures. Each respective portion of the example training flow can be performed by any (or any combination) of one or more computing devices. Moreover, one or more portion(s) of the example training flow can be implemented on the hardware components of the device(s) described herein, for example, to train one or more systems or models. FIG. 8 depicts elements performed in a particular order for purposes of illustration and discussion. Those of ordinary skill in the art, using the disclosures provided herein, will understand that the elements of any of the methods discussed herein can be adapted, rearranged, expanded, omitted, combined, or modified in various ways without deviating from the scope of the present disclosure. FIG. 8 is described with reference to elements/terms described with respect to other systems and figures for exemplary illustrated purposes and is not meant to be limiting. One or more portions of the example training flow can be performed additionally, or alternatively, by other systems.

[0152] Initially, development model 16 can persist in an initial state as an initialized model 21. Development model 16 can be initialized with weight values. Initial weight values can be random or based on an initialization schema. Initial weight values can be based on prior pre-training for the same or for a different model.

[0153] Initialized model 21 can undergo pre-training in a pre-training stage 22. Pre-training stage 22 can be implemented using one or more pre-training pipelines 17-2 over data from dataset(s) 17-1. Pre-training can be omitted, for example, if initialized model 21 is already pre-trained (e.g., development model 16 contains, is, or is based on a pre-trained foundational model or an expert model).

[0154] Pre-trained model 23 can then be a new version of development model 16, which can persist as development model 16 or as a new development model. Pre-trained model 23 can be the initial state if development model 16 was already pre-trained. Pre-trained model 23 can undergo fine-tuning in a fine-tuning stage 24. Fine-tuning stage 24 can be implemented using one or more fine-tuning pipelines 17-3 over data from dataset(s) 17-1. Fine-tuning can be omitted, for example, if a pre-trained model as satisfactory performance, if the model was already fine-tuned, or if other tuning approaches are preferred.

[0155] Fine-tuned model 29 can then be a new version of development model 16, which can persist as development model 16 or as a new development model. Fine-tuned model 29 can be the initial state if development model 16 was already fine-tuned. Fine-tuned model 29 can undergo refinement with user feedback 26. For instance, refinement with user feedback 26 can include reinforcement learning, optionally based on human feedback from human users of fine-tuned model 25. As reinforcement learning can be a form of fine-tuning, it is to be understood that fine-tuning stage 24 can subsume the stage for refining with user feedback 26. Refinement with user feedback 26 can produce a refined model 27. Refined model 27 can be output to downstream system(s) 28 for deployment or further development.

[0156] In some implementations, computational optimization operations can be applied before, during, or after each stage. For instance, initialized model 21 can undergo computational optimization 29-1 (e.g., using computational optimization toolkit 19) before pre-training stage 22. Pre-trained model 23 can undergo computational optimization 29-2 (e.g., using computational optimization toolkit 19) before fine-tuning stage 24. Fine-tuned model 25 can undergo computational optimization 29-3 (e.g., using computational optimization toolkit 19) before refinement with user feedback 26. Refined model 27 can undergo computational optimization 29-4 (e.g., using computational optimization toolkit 19) before output to downstream system(s) 28. Computational optimization(s) 29-1, . . . , 29-4 can all be the same, all be different, or include at least some different optimization techniques.

Example Machine-Learned Model Inference System

[0157] Figure 9 is a block diagram of an inference system for operating one or more machine-learned model(s) 1 to perform inference (e.g., for training, for deployment, etc.). A model host 31 can receive machine-learned model(s) 1. Model host 31 can host one or more model instance(s) 31-1, which can be one or multiple instances of one or multiple models. Model host 31 can host model instance(s) 31-1 using available compute resources 31-2 associated with model host 31.

[0158] Model host 31 can perform inference on behalf of one or more client(s) 32. Client(s) 32 can transmit an input request 33 to model host 31. Using input request 33, model host 31 can obtain input(s) 2 for input to machine-learned model(s) 1. Machine-learned model(s) 1 can process input(s) 2 to generate output(s) 3. Using output(s) 3, model host 31 can return an output payload 34 for responding to input request 33 from client(s) 32. Output payload 34 can include or be based on output(s) 3.

[0159] Model host 31 can leverage various other resources and tools to augment the inference task. For instance, model host 31 can communicate with tool interfaces 35 to facilitate tool use by model instance(s) 31-1. Tool interfaces 35 can include local or remote APIs. Tool interfaces 35 can include integrated scripts or other software functionality. Model host 31 can engage online learning interface(s) 36 to facilitate ongoing improvements to machine-learned model(s) 1. For instance, online learning interface(s) 36 can be used within reinforcement learning loops to retrieve user feedback on inferences served by model host 31. Model host 31 can access runtime data source(s) 37 for augmenting input(s) 2 with additional contextual information. For instance, runtime data source(s) 37 can include a knowledge graph 37-1 that facilitates structured information retrieval for information associated with input request(s) 33 (e.g., a search engine service). Runtime data source(s) 37 can include public or private, external or local database(s) 37-2 that can store information associated with input request(s) 33 for augmenting input(s) 2. Runtime data source(s) 37 can include account data 37-3 which can be retrieved in association with a user account corresponding to a client 32 for customizing the behavior of model host 31 accordingly.

[0160] Model host 31 can be implemented by one or multiple computing devices or systems. Client(s) 2 can be implemented by one or multiple computing devices or systems, which can include computing devices or systems shared with model host 31.

[0161] For example, model host 31 can operate on a server system that provides a machine-learning service to client device(s) that operate client(s) 32 (e.g., over a local or wide-area network). Client device(s) can be end-user devices used by individuals. Client

device(s) can be server systems that operate client(s) 32 to provide various functionality as a service to downstream end-user devices.

[0162] In some implementations, model host 31 can operate on a same device or system as client(s) 32. Model host 31 can be a machine-learning service that runs on-device to provide machine-learning functionality to one or multiple applications operating on a client device, which can include an application implementing client(s) 32. Model host 31 can be a part of a same application as client(s) 32. For instance, model host 31 can be a subroutine or method implemented by one part of an application, and client(s) 32 can be another subroutine or method that engages model host 31 to perform inference functions within the application. It is to be understood that model host 31 and client(s) 32 can have various different configurations.

[0163] Model instance(s) 31-1 can include one or more machine-learned models that are available for performing inference. Model instance(s) 31-1 can include weights or other model components that are stored on in persistent storage, temporarily cached, or loaded into high-speed memory. Model instance(s) 31-1 can include multiple instance(s) of the same model (e.g., for parallel execution of more requests on the same model). Model instance(s) 31-1 can include instance(s) of different model(s). Model instance(s) 31-1 can include cached intermediate states of active or inactive model(s) used to accelerate inference of those models. For instance, an inference session with a particular model may generate significant amounts of computational results that can be re-used for future inference runs (e.g., using a KV cache for transformer-based models). These computational results can be saved in association with that inference session so that session can be executed more efficiently when resumed.

[0164] Compute resource(s) 31-2 can include one or more processors (central processing units, graphical processing units, tensor processing units, machine-learning accelerators, etc.) connected to one or more memory devices. Compute resource(s) 31-2 can include a dynamic pool of available resources shared with other processes. Compute resource(s) 31-2 can include memory devices large enough to fit an entire model instance in a single memory instance. Compute resource(s) 31-2 can also shard model instance(s) across multiple memory devices (e.g., using data parallelization or tensor parallelization, etc.). This can be done to increase parallelization or to execute a large model using multiple memory devices which individually might not be able to fit the entire model into memory.

[0165] Input request 33 can include data for input(s) 2. Model host 31 can process input request 33 to obtain input(s) 2. Input(s) 2 can be obtained directly from input request 33 or

can be retrieved using input request 33. Input request 33 can be submitted to model host 31 via an API.

[0166] Model host 31 can perform inference over batches of input requests 33 in parallel. For instance, a model instance 31-1 can be configured with an input structure that has a batch dimension. Separate input(s) 2 can be distributed across the batch dimension (e.g., rows of an array). The separate input(s) 2 can include completely different contexts. The separate input(s) 2 can be multiple inference steps of the same task. The separate input(s) 2 can be staggered in an input structure, such that any given inference cycle can be operating on different portions of the respective input(s) 2. In this manner, for instance, model host 31 can perform inference on the batch in parallel, such that output(s) 3 can also contain the batch dimension and return the inference results for the batched input(s) 2 in parallel. In this manner, for instance, batches of input request(s) 33 can be processed in parallel for higher throughput of output payload(s) 34.

[0167] Output payload 34 can include or be based on output(s) 3 from machine-learned model(s) 1. Model host 31 can process output(s) 3 to obtain output payload 34. This can include chaining multiple rounds of inference (e.g., iteratively, recursively, across the same model(s) or different model(s)) to arrive at a final output for a task to be returned in output payload 34. Output payload 34 can be transmitted to client(s) 32 via an API.

[0168] Online learning interface(s) 36 can facilitate reinforcement learning of machine-learned model(s) 1. Online learning interface(s) 36 can facilitate reinforcement learning with human feedback (RLHF). Online learning interface(s) 36 can facilitate federated learning of machine-learned model(s) 1.

[0169] Model host 31 can execute machine-learned model(s) 1 to perform inference for various tasks using various types of data. For example, various different input(s) 2 and output(s) 3 can be used for various different tasks. In some implementations, input(s) 2 can be or otherwise represent image data. Machine-learned model(s) 1 can process the image data to generate an output. As an example, machine-learned model(s) 1 can process the image data to generate an image recognition output (e.g., a recognition of the image data, a latent embedding of the image data, an encoded representation of the image data, a hash of the image data, etc.). As another example, machine-learned model(s) 1 can process the image data to generate an image segmentation output. As another example, machine-learned model(s) 1 can process the image data to generate an image classification output. As another example, machine-learned model(s) 1 can process the image data to generate an image data modification output (e.g., an alteration of the image data, etc.). As another example, machine-

learned model(s) 1 can process the image data to generate an encoded image data output (e.g., an encoded and/or compressed representation of the image data, etc.). As another example, machine-learned model(s) 1 can process the image data to generate an upscaled image data output. As another example, machine-learned model(s) 1 can process the image data to generate a prediction output.

[0170] In some implementations, the task is a computer vision task. In some cases, input(s) 2 includes pixel data for one or more images and the task is an image processing task. For example, the image processing task can be image classification, where the output is a set of scores, each score corresponding to a different object class and representing the likelihood that the one or more images depict an object belonging to the object class. The image processing task may be object detection, where the image processing output identifies one or more regions in the one or more images and, for each region, a likelihood that region depicts an object of interest. As another example, the image processing task can be image segmentation, where the image processing output defines, for each pixel in the one or more images, a respective likelihood for each category in a predetermined set of categories. For example, the set of categories can be foreground and background. As another example, the set of categories can be object classes. As another example, the image processing task can be depth estimation, where the image processing output defines, for each pixel in the one or more images, a respective depth value. As another example, the image processing task can be motion estimation, where the network input includes multiple images, and the image processing output defines, for each pixel of one of the input images, a motion of the scene depicted at the pixel between the images in the network input.

[0171] In some implementations, input(s) 2 can be or otherwise represent natural language data. Machine-learned model(s) 1 can process the natural language data to generate an output. As an example, machine-learned model(s) 1 can process the natural language data to generate a language encoding output. As another example, machine-learned model(s) 1 can process the natural language data to generate a latent text embedding output. As another example, machine-learned model(s) 1 can process the natural language data to generate a translation output. As another example, machine-learned model(s) 1 can process the natural language data to generate a classification output. As another example, machine-learned model(s) 1 can process the natural language data to generate a textual segmentation output. As another example, machine-learned model(s) 1 can process the natural language data to generate a semantic intent output. As another example, machine-learned model(s) 1 can process the natural language data to generate an upscaled text or natural language output

(e.g., text or natural language data that is higher quality than the input text or natural language, etc.). As another example, machine-learned model(s) 1 can process the natural language data to generate a prediction output (e.g., one or more predicted next portions of natural language content).

[0172] In some implementations, input(s) 2 can be or otherwise represent speech data (e.g., data describing spoken natural language, such as audio data, textual data, etc.). Machine-learned model(s) 1 can process the speech data to generate an output. As an example, machine-learned model(s) 1 can process the speech data to generate a speech recognition output. As another example, machine-learned model(s) 1 can process the speech data to generate a speech translation output. As another example, machine-learned model(s) 1 can process the speech data to generate a latent embedding output. As another example, machine-learned model(s) 1 can process the speech data to generate an encoded speech output (e.g., an encoded and/or compressed representation of the speech data, etc.). As another example, machine-learned model(s) 1 can process the speech data to generate an upscaled speech output (e.g., speech data that is higher quality than the input speech data, etc.). As another example, machine-learned model(s) 1 can process the speech data to generate a textual representation output (e.g., a textual representation of the input speech data, etc.). As another example, machine-learned model(s) 1 can process the speech data to generate a prediction output.

[0173] In some implementations, input(s) 2 can be or otherwise represent latent encoding data (e.g., a latent space representation of an input, etc.). Machine-learned model(s) 1 can process the latent encoding data to generate an output. As an example, machine-learned model(s) 1 can process the latent encoding data to generate a recognition output. As another example, machine-learned model(s) 1 can process the latent encoding data to generate a reconstruction output. As another example, machine-learned model(s) 1 can process the latent encoding data to generate a search output. As another example, machine-learned model(s) 1 can process the latent encoding data to generate a reclustering output. As another example, machine-learned model(s) 1 can process the latent encoding data to generate a prediction output.

[0174] In some implementations, input(s) 2 can be or otherwise represent statistical data. Statistical data can be, represent, or otherwise include data computed and/or calculated from some other data source. Machine-learned model(s) 1 can process the statistical data to generate an output. As an example, machine-learned model(s) 1 can process the statistical data to generate a recognition output. As another example, machine-learned model(s) 1 can

process the statistical data to generate a prediction output. As another example, machine-learned model(s) 1 can process the statistical data to generate a classification output. As another example, machine-learned model(s) 1 can process the statistical data to generate a segmentation output. As another example, machine-learned model(s) 1 can process the statistical data to generate a visualization output. As another example, machine-learned model(s) 1 can process the statistical data to generate a diagnostic output.

[0175] In some implementations, input(s) 2 can be or otherwise represent sensor data. Machine-learned model(s) 1 can process the sensor data to generate an output. As an example, machine-learned model(s) 1 can process the sensor data to generate a recognition output. As another example, machine-learned model(s) 1 can process the sensor data to generate a prediction output. As another example, machine-learned model(s) 1 can process the sensor data to generate a classification output. As another example, machine-learned model(s) 1 can process the sensor data to generate a segmentation output. As another example, machine-learned model(s) 1 can process the sensor data to generate a visualization output. As another example, machine-learned model(s) 1 can process the sensor data to generate a diagnostic output. As another example, machine-learned model(s) 1 can process the sensor data to generate a detection output.

[0176] In some implementations, machine-learned model(s) 1 can be configured to perform a task that includes encoding input data for reliable and/or efficient transmission or storage (and/or corresponding decoding). For example, the task may be an audio compression task. The input may include audio data and the output may comprise compressed audio data. In another example, the input includes visual data (e.g. one or more images or videos), the output comprises compressed visual data, and the task is a visual data compression task. In another example, the task may comprise generating an embedding for input data (e.g. input audio or visual data). In some cases, the input includes audio data representing a spoken utterance and the task is a speech recognition task. The output may comprise a text output which is mapped to the spoken utterance. In some cases, the task comprises encrypting or decrypting input data. In some cases, the task comprises a microprocessor performance task, such as branch prediction or memory address translation.

[0177] In some implementations, the task is a generative task, and machine-learned model(s) 1 can be configured to output content generated in view of input(s) 2. For instance, input(s) 2 can be or otherwise represent data of one or more modalities that encodes context for generating additional content.

[0178] In some implementations, the task can be a text completion task. Machine-learned model(s) 1 can be configured to process input(s) 2 that represent textual data and to generate output(s) 3 that represent additional textual data that completes a textual sequence that includes input(s) 2. For instance, machine-learned model(s) 1 can be configured to generate output(s) 3 to complete a sentence, paragraph, or portion of text that follows from a portion of text represented by input(s) 2.

[0179] In some implementations, the task can be an instruction following task. Machine-learned model(s) 1 can be configured to process input(s) 2 that represent instructions to perform a function and to generate output(s) 3 that advance a goal of satisfying the instruction function (e.g., at least a step of a multi-step procedure to perform the function). Output(s) 3 can represent data of the same or of a different modality as input(s) 2. For instance, input(s) 2 can represent textual data (e.g., natural language instructions for a task to be performed) and machine-learned model(s) 1 can process input(s) 2 to generate output(s) 3 that represent textual data responsive to the instructions (e.g., natural language responses, programming language responses, machine language responses, etc.). Input(s) 2 can represent image data (e.g., image-based instructions for a task to be performed, optionally accompanied by textual instructions) and machine-learned model(s) 1 can process input(s) 2 to generate output(s) 3 that represent textual data responsive to the instructions (e.g., natural language responses, programming language responses, machine language responses, etc.). One or more output(s) 3 can be iteratively or recursively generated to sequentially process and accomplish steps toward accomplishing the requested functionality. For instance, an initial output can be executed by an external system or be processed by machine-learned model(s) 1 to complete an initial step of performing a function. Multiple steps can be performed, with a final output being obtained that is responsive to the initial instructions.

[0180] In some implementations, the task can be a question answering task. Machine-learned model(s) 1 can be configured to process input(s) 2 that represent a question to answer and to generate output(s) 3 that advance a goal of returning an answer to the question (e.g., at least a step of a multi-step procedure to perform the function). Output(s) 3 can represent data of the same or of a different modality as input(s) 2. For instance, input(s) 2 can represent textual data (e.g., natural language instructions for a task to be performed) and machine-learned model(s) 1 can process input(s) 2 to generate output(s) 3 that represent textual data responsive to the question (e.g., natural language responses, programming language responses, machine language responses, etc.). Input(s) 2 can represent image data (e.g., image-based instructions for a task to be performed, optionally accompanied by textual

instructions) and machine-learned model(s) 1 can process input(s) 2 to generate output(s) 3 that represent textual data responsive to the question (e.g., natural language responses, programming language responses, machine language responses, etc.). One or more output(s) 3 can be iteratively or recursively generated to sequentially process and accomplish steps toward answering the question. For instance, an initial output can be executed by an external system or be processed by machine-learned model(s) 1 to complete an initial step of obtaining an answer to the question (e.g., querying a database, performing a computation, executing a script, etc.). Multiple steps can be performed, with a final output being obtained that is responsive to the question.

[0181] In some implementations, the task can be an image generation task. Machine-learned model(s) 1 can be configured to process input(s) 2 that represent context regarding a desired portion of image content. The context can include text data, image data, audio data, etc. Machine-learned model(s) 1 can be configured to generate output(s) 3 that represent image data that depicts imagery related to the context. For instance, machine-learned model(s) 1 can be configured to generate pixel data of an image. Values for channel(s) associated with the pixels in the pixel data can be selected based on the context (e.g., based on a probability determined based on the context).

[0182] In some implementations, the task can be an audio generation task. Machine-learned model(s) 1 can be configured to process input(s) 2 that represent context regarding a desired portion of audio content. The context can include text data, image data, audio data, etc. Machine-learned model(s) 1 can be configured to generate output(s) 3 that represent audio data related to the context. For instance, machine-learned model(s) 1 can be configured to generate waveform data in the form of an image (e.g., a spectrogram). Values for channel(s) associated with pixels of the image can be selected based on the context. Machine-learned model(s) 1 can be configured to generate waveform data in the form of a sequence of discrete samples of a continuous waveform. Values of the sequence can be selected based on the context (e.g., based on a probability determined based on the context).

[0183] In some implementations, the task can be a data generation task. Machine-learned model(s) 1 can be configured to process input(s) 2 that represent context regarding a desired portion of data (e.g., data from various data domains, such as sensor data, image data, multimodal data, statistical data, etc.). The desired data can be, for instance, synthetic data for training other machine-learned models. The context can include arbitrary data type(s). Machine-learned model(s) 1 can be configured to generate output(s) 3 that represent data that aligns with the desired data. For instance, machine-learned model(s) 1 can be configured to

generate data values for populating a dataset. Values for the data object(s) can be selected based on the context (e.g., based on a probability determined based on the context).

Example Computing Systems and Devices

[0184] Figure 10 is a block diagram of an example networked computing system that can perform aspects of example implementations of the present disclosure. The system can include a number of computing devices and systems that are communicatively coupled over a network 49. An example computing device 50 is described to provide an example of a computing device that can perform any aspect of the present disclosure (e.g., implementing model host 31, client(s) 32, or both). An example server computing system 60 is described as an example of a server computing system that can perform any aspect of the present disclosure (e.g., implementing model host 31, client(s) 32, or both). Computing device 50 and server computing system(s) 60 can cooperatively interact (e.g., over network 49) to perform any aspect of the present disclosure (e.g., implementing model host 31, client(s) 32, or both). Model development platform system 70 is an example system that can host or serve model development platform(s) 12 for development of machine-learned models. Third-party system(s) 80 are example system(s) with which any of computing device 50, server computing system(s) 60, or model development platform system(s) 70 can interact in the performance of various aspects of the present disclosure (e.g., engaging third-party tools, accessing third-party databases or other resources, etc.).

[0185] Network 49 can be any type of communications network, such as a local area network (e.g., intranet), wide area network (e.g., Internet), or some combination thereof and can include any number of wired or wireless links. In general, communication over network 49 can be carried via any type of wired or wireless connection, using a wide variety of communication protocols (e.g., TCP/IP, HTTP, SMTP, FTP), encodings or formats (e.g., HTML, XML), or protection schemes (e.g., VPN, secure HTTP, SSL). Network 49 can also be implemented via a system bus. For instance, one or more devices or systems of Figure 10 can be co-located with, contained by, or otherwise integrated into one or more other devices or systems.

[0186] Computing device 50 can be any type of computing device, such as, for example, a personal computing device (e.g., laptop or desktop), a mobile computing device (e.g., smartphone or tablet), a gaming console or controller, a wearable computing device, an embedded computing device, a server computing device, a virtual machine operating on a host device, or any other type of computing device. Computing device 50 can be a client

computing device. Computing device 50 can be an end-user computing device. Computing device 50 can be a computing device of a service provided that provides a service to an end user (who may use another computing device to interact with computing device 50).

[0187] Computing device 50 can include one or more processors 51 and a memory 52. Processor(s) 51 can be any suitable processing device (e.g., a processor core, a microprocessor, an ASIC, an FPGA, a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. Memory 52 can include one or more non-transitory computer-readable storage media, such as HBM, RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. Memory 52 can store data 53 and instructions 54 which can be executed by processor(s) 51 to cause computing device 50 to perform operations. The operations can implement any one or multiple features described herein. The operations can implement example methods and techniques described herein.

[0188] Computing device 50 can also include one or more input components that receive user input. For example, a user input component can be a touch-sensitive component (e.g., a touch-sensitive display screen or a touch pad) that is sensitive to the touch of a user input object (e.g., a finger or a stylus). The touch-sensitive component can serve to implement a virtual keyboard. Other example user input components include a microphone, camera, LIDAR, a physical keyboard or other buttons, or other means by which a user can provide user input.

[0189] Computing device 50 can store or include one or more machine-learned models 55. Machine-learned models 55 can include one or more machine-learned model(s) 1, such as a sequence processing model 4. Machine-learned models 55 can include one or multiple model instance(s) 31-1. Machine-learned model(s) 55 can be received from server computing system(s) 60, model development platform system 70, third party system(s) 80 (e.g., an application distribution platform), or developed locally on computing device 50. Machine-learned model(s) 55 can be loaded into memory 52 and used or otherwise implemented by processor(s) 51. Computing device 50 can implement multiple parallel instances of machine-learned model(s) 55.

[0190] Server computing system(s) 60 can include one or more processors 61 and a memory 62. Processor(s) 61 can be any suitable processing device (e.g., a processor core, a microprocessor, an ASIC, an FPGA, a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. Memory 62 can include one or more non-transitory computer-readable storage media, such as HBM, RAM, ROM,

EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. Memory 62 can store data 63 and instructions 64 which can be executed by processor(s) 61 to cause server computing system(s) 60 to perform operations. The operations can implement any one or multiple features described herein. The operations can implement example methods and techniques described herein.

[0191] In some implementations, server computing system 60 includes or is otherwise implemented by one or multiple server computing devices. In instances in which server computing system 60 includes multiple server computing devices, such server computing devices can operate according to sequential computing architectures, parallel computing architectures, or some combination thereof.

[0192] Server computing system 60 can store or otherwise include one or more machine-learned models 65. Machine-learned model(s) 65 can be the same as or different from machine-learned model(s) 55. Machine-learned models 65 can include one or more machine-learned model(s) 1, such as a sequence processing model 4. Machine-learned models 65 can include one or multiple model instance(s) 31-1. Machine-learned model(s) 65 can be received from computing device 50, model development platform system 70, third party system(s) 80, or developed locally on server computing system(s) 60. Machine-learned model(s) 65 can be loaded into memory 62 and used or otherwise implemented by processor(s) 61. Server computing system(s) 60 can implement multiple parallel instances of machine-learned model(s) 65.

[0193] In an example configuration, machine-learned models 65 can be included in or otherwise stored and implemented by server computing system 60 to establish a client-server relationship with computing device 50 for serving model inferences. For instance, server computing system(s) 60 can implement model host 31 on behalf of client(s) 32 on computing device 50. For instance, machine-learned models 65 can be implemented by server computing system 60 as a portion of a web service (e.g., remote machine-learned model hosting service, such as an online interface for performing machine-learned model operations over a network on server computing system(s) 60). For instance, server computing system(s) 60 can communicate with computing device 50 over a local intranet or internet connection. For instance, computing device 50 can be a workstation or endpoint in communication with server computing system(s) 60, with implementation of machine-learned models 65 being managed by server computing system(s) 60 to remotely perform inference (e.g., for runtime or training operations), with output(s) returned (e.g., cast, streamed, etc.) to computing device

50. Machine-learned models 65 can work cooperatively or interoperatively with machine-learned models 55 on computing device 50 to perform various tasks.

[0194] Model development platform system(s) 70 can include one or more processors 71 and a memory 72. Processor(s) 71 can be any suitable processing device (e.g., a processor core, a microprocessor, an ASIC, an FPGA, a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. Memory 72 can include one or more non-transitory computer-readable storage media, such as HBM, RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. Memory 72 can store data 73 and instructions 74 which can be executed by processor(s) 71 to cause model development platform system(s) 70 to perform operations. The operations can implement any one or multiple features described herein. The operations can implement example methods and techniques described herein. Example operations include the functionality described herein with respect to model development platform 12. This and other functionality can be implemented by developer tool(s) 75.

[0195] Third-party system(s) 80 can include one or more processors 81 and a memory 82. Processor(s) 81 can be any suitable processing device (e.g., a processor core, a microprocessor, an ASIC, an FPGA, a controller, a microcontroller, etc.) and can be one processor or a plurality of processors that are operatively connected. Memory 82 can include one or more non-transitory computer-readable storage media, such as HBM, RAM, ROM, EEPROM, EPROM, flash memory devices, magnetic disks, etc., and combinations thereof. Memory 82 can store data 83 and instructions 84 which can be executed by processor(s) 81 to cause third-party system(s) 80 to perform operations. The operations can implement any one or multiple features described herein. The operations can implement example methods and techniques described herein. Example operations include the functionality described herein with respect to tools and other external resources called when training or performing inference with machine-learned model(s) 1, 4, 16, 20, 55, 65, etc. (e.g., third-party resource(s) 85).

[0196] Figure 10 illustrates one example arrangement of computing systems that can be used to implement the present disclosure. Other computing system configurations can be used as well. For example, in some implementations, one or both of computing system 50 or server computing system(s) 60 can implement all or a portion of the operations of model development platform system 70. For example, computing system 50 or server computing system(s) 60 can implement developer tool(s) 75 (or extensions thereof) to develop, update/train, or refine machine-learned models 1, 4, 16, 20, 55, 65, etc. using one or more

techniques described herein with respect to model alignment toolkit 17. In this manner, for instance, computing system 50 or server computing system(s) 60 can develop, update/train, or refine machine-learned models based on local datasets (e.g., for model personalization/customization, as permitted by user data preference selections).

[0197] Figure 11 is a block diagram of an example computing device 98 that performs according to example embodiments of the present disclosure. Computing device 98 can be a user computing device or a server computing device (e.g., computing device 50, server computing system(s) 60, etc.). Computing device 98 can implement model host 31. For instance, computing device 98 can include a number of applications (e.g., applications 1 through N). Each application can contain its own machine learning library and machine-learned model(s). For example, each application can include a machine-learned model. Example applications include a text messaging application, an email application, a dictation application, a virtual keyboard application, a browser application, etc. As illustrated in Figure 11, each application can communicate with a number of other components of the computing device, such as, for example, one or more sensors, a context manager, a device state component, or additional components. In some implementations, each application can communicate with each device component using an API (e.g., a public API). In some implementations, the API used by each application is specific to that application.

[0198] Figure 12 is a block diagram of an example computing device 99 that performs according to example embodiments of the present disclosure. Computing device 99 can be the same as or different from computing device 98. Computing device 99 can be a user computing device or a server computing device (e.g., computing device 50, server computing system(s) 60, etc.). Computing device 99 can implement model host 31. For instance, computing device 99 can include a number of applications (e.g., applications 1 through N). Each application can be in communication with a central intelligence layer. Example applications include a text messaging application, an email application, a dictation application, a virtual keyboard application, a browser application, etc. In some implementations, each application can communicate with the central intelligence layer (and model(s) stored therein) using an API (e.g., a common API across all applications).

[0199] The central intelligence layer can include a number of machine-learned models. For example, as illustrated in Figure 12, a respective machine-learned model can be provided for each application and managed by the central intelligence layer. In other implementations, two or more applications can share a single machine-learned model. For example, in some implementations, the central intelligence layer can provide a single model for all of the

applications. In some implementations, the central intelligence layer is included within or otherwise implemented by an operating system of computing device 99.

[0200] The central intelligence layer can communicate with a central device data layer. The central device data layer can be a centralized repository of data for computing device 99. As illustrated in Figure 12, the central device data layer can communicate with a number of other components of the computing device, such as, for example, one or more sensors, a context manager, a device state component, or additional components. In some implementations, the central device data layer can communicate with each device component using an API (e.g., a private API).

Additional Disclosure

[0201] The technology discussed herein makes reference to servers, databases, software applications, and other computer-based systems, as well as actions taken and information sent to and from such systems. The inherent flexibility of computer-based systems allows for a great variety of possible configurations, combinations, and divisions of tasks and functionality between and among components. For instance, processes discussed herein can be implemented using a single device or component or multiple devices or components working in combination. Databases and applications can be implemented on a single system or distributed across multiple systems. Distributed components can operate sequentially or in parallel.

[0202] While the present subject matter has been described in detail with respect to various specific example embodiments thereof, each example is provided by way of explanation, not limitation of the disclosure. Those skilled in the art, upon attaining an understanding of the foregoing, can readily produce alterations to, variations of, and equivalents to such embodiments. Accordingly, the subject disclosure does not preclude inclusion of such modifications, variations or additions to the present subject matter as would be readily apparent to one of ordinary skill in the art. For instance, features illustrated or described as part of one embodiment can be used with another embodiment to yield a still further embodiment. Thus, it is intended that the present disclosure cover such alterations, variations, and equivalents.

[0203] Aspects of the disclosure have been described in terms of illustrative embodiments thereof. Any and all features in the following claims can be combined or rearranged in any way possible, including combinations of claims not explicitly enumerated in combination together, as the example claim dependencies listed herein should not be read

as limiting the scope of possible combinations of features disclosed herein. Accordingly, the scope of the present disclosure is by way of example rather than by way of limitation, and the subject disclosure does not preclude inclusion of such modifications, variations or additions to the present subject matter as would be readily apparent to one of ordinary skill in the art. Moreover, terms are described herein using lists of example elements joined by conjunctions such as “and,” “or,” “but,” etc. It should be understood that such conjunctions are provided for explanatory purposes only. Clauses and other sequences of items joined by a particular conjunction such as “or,” for example, can refer to “and/or,” “at least one of,” “any combination of” example elements listed therein, etc. Terms such as “based on” should be understood as “based at least in part on.”

[0204] The term “can” should be understood as referring to a possibility of a feature in various implementations and not as prescribing an ability that is necessarily present in every implementation. For example, the phrase “X can perform Y” should be understood as indicating that, in various implementations, X has the potential to be configured to perform Y, and not as indicating that in every instance X must always be able to perform Y. It should be understood that, in various implementations, X might be unable to perform Y and remain within the scope of the present disclosure.

[0205] The term “may” should be understood as referring to a possibility of a feature in various implementations and not as prescribing an ability that is necessarily present in every implementation. For example, the phrase “X may perform Y” should be understood as indicating that, in various implementations, X has the potential to be configured to perform Y, and not as indicating that in every instance X must always be able to perform Y. It should be understood that, in various implementations, X might be unable to perform Y and remain within the scope of the present disclosure.

WHAT IS CLAIMED IS:

1. A computer-implemented method for automating tool use based on learned similarity, the method comprising:
 - obtaining, by a computing system comprising one or more computing devices, data descriptive of a query;
 - generating, by the computing system, a query embedding for the query, wherein the query embedding is expressed within a latent embedding space;
 - performing, by the computing system, a similarity search for the query embedding within the latent embedding space to identify one or more previously-defined embeddings associated with one or more previously-defined query-response pairs;
 - generating, by the computing system, a prompt based on the query and the one or more previously-defined query-response pairs;
 - providing, by the computing system, the prompt as an input for processing by a machine-learned language model; and
 - receiving, by the computing system, a model response to the query that was output by the machine-learned language model based on processing of the prompt.
2. The computer-implemented method of any preceding claim, wherein the model response comprises one or more tokens that, when executed by the computing system, causes a structural tool to retrieve or transform information.
3. The computer-implemented method of claim 2, further comprising:
 - processing, by the computing system, the model response to generate a system response;
 - wherein processing the model response comprises executing the one or more tokens to cause the structural tool to retrieve or transform the information; and
 - wherein the system response is based at least in part on the information retrieved or transformed by the structural tool responsive to execution of the one or more tokens.

4. The computer-implemented method of claim 3, wherein the query comprises a natural language question and the system response comprises a natural language answer to the natural language question.

5. The computer-implemented method of any of claims 2-4, wherein the structural tool comprises a programming language interpreter configured to execute instructions expressed in a computer language.

6. The computer-implemented method of claim 5, further comprising:
pre-loading, by the computing system, one or more libraries or datasets associated with the computer language into a computer environment associated with the programming language interpreter.

7. The computer-implemented method of claim 5 or 6, wherein the computer language comprises Python computer language.

8. The computer-implemented method of any of claims 2-7, wherein the structural tool comprises:

a database lookup to access additional information from a database;
an application programming interface (API) call to request and receive additional information via the API;
a query service that queries results from a search engine, knowledge graph, or digital assistant.

9. The computer-implemented method of any preceding claim, wherein generating, by the computing system, the prompt based on the query and the one or more previously-defined query-response pairs comprises concatenating the query with the one or more previously-defined query-response pairs.

10. The computer-implemented method of any preceding claim, wherein generating, by the computing system, the prompt based on the query and the one or more previously-

defined query-response pairs comprises ranking the one or more previously-defined query-response pairs.

11. The computer-implemented method of any preceding claim, wherein:
at least one of the one previously-defined query-response pairs demonstrates usage of a structural tool; and
processing of the prompt by the machine-learned language model comprises usage of the structural tool.

12. The computer-implemented method of any preceding claim, further comprising, prior to obtaining the data descriptive of the query:
generating, by the computing system, the previously-defined embedding for each of the one or more previously-defined query-response pairs based at least in part on a query portion of the previously-defined query-response pair.

13. The computer-implemented method of any preceding claim, wherein performing, by the computing system, the similarity search for the query embedding within the latent embedding space comprises identifying a top- k set of previously-defined embeddings from the embedding space, wherein k is a hyperparameter.

14. The computer-implemented method of any preceding claim, wherein the model response comprises an executable program.

15. The computer-implemented method of any preceding claim, wherein the machine-learned language model comprises a pre-trained large language model.

16. The computer-implemented method of any preceding claim, further comprising:
fine-tuning the machine-learned language model on the previously-defined query-response pairs.

17. The computer-implemented method of any preceding claim, wherein providing, by the computing system, the prompt as the input for processing by the machine-learned

language model comprises running, by the computing system, the machine-learned language model to process the prompt.

18. A computer system configured to perform the method of any preceding claim.

19. The computer system of claim 18, wherein the computer system comprises the machine-learned language model.

20. One or more non-transitory computer-readable media that store instructions for performing the method of any of claims 1-17.

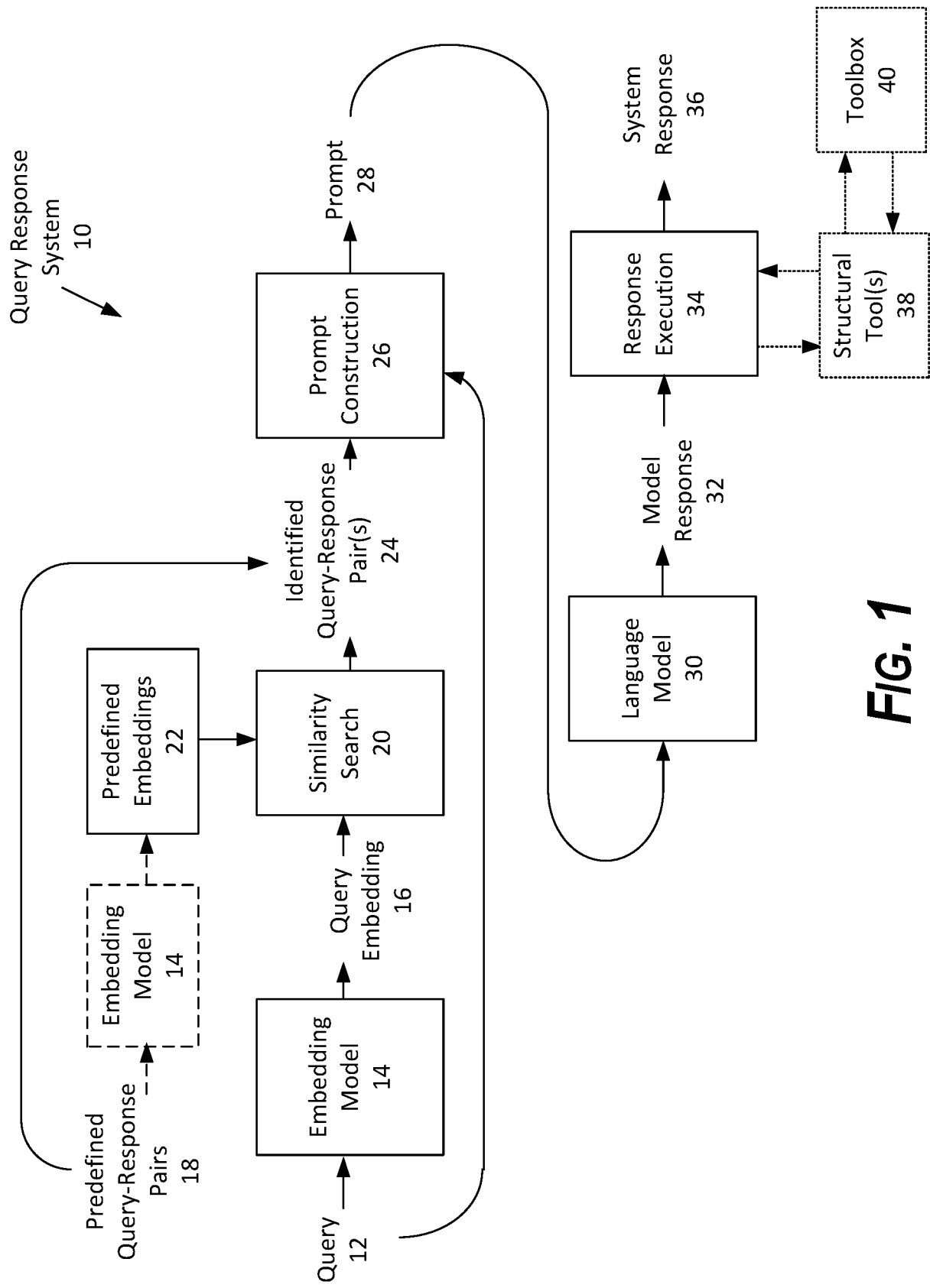


FIG. 1

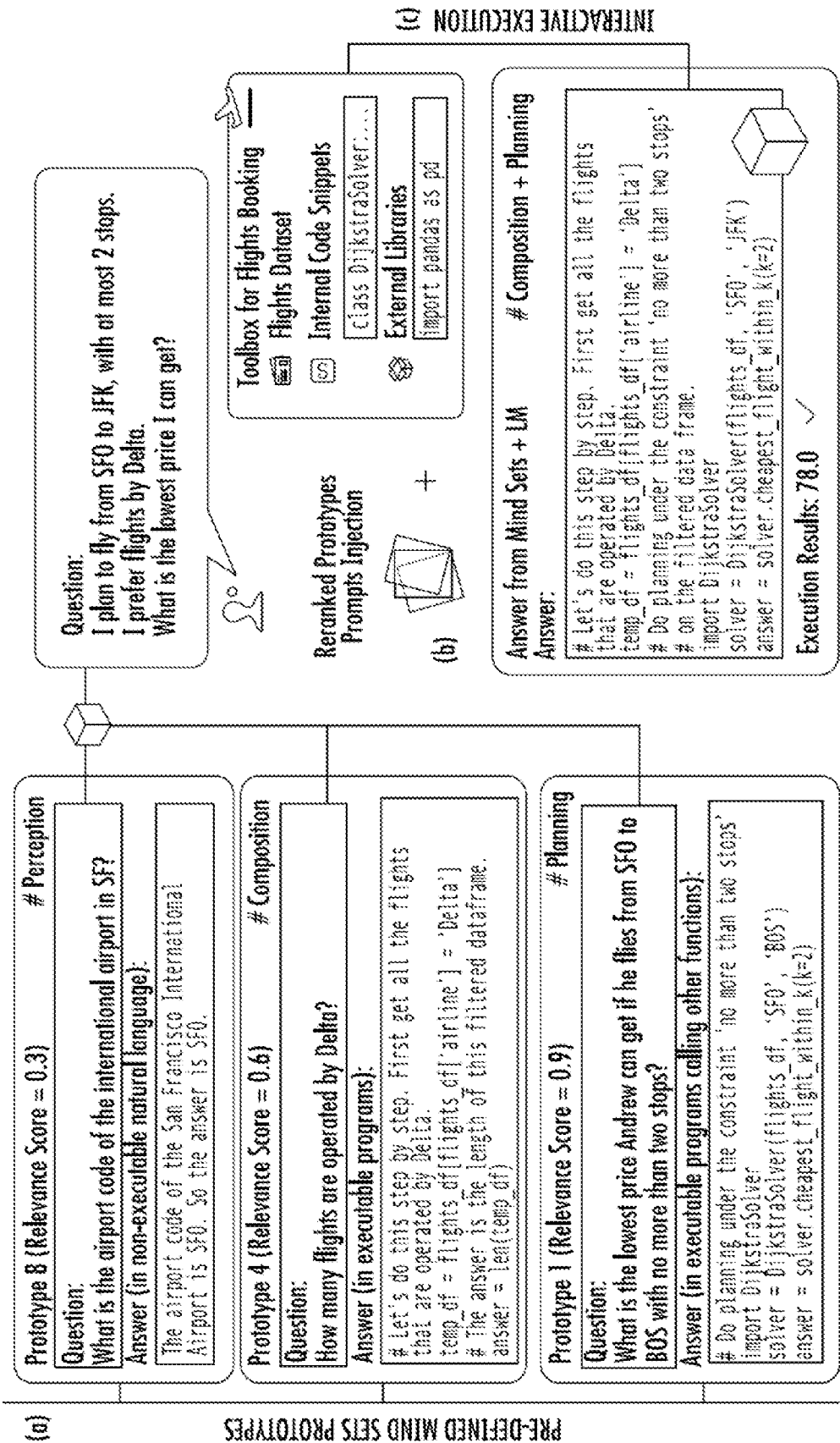
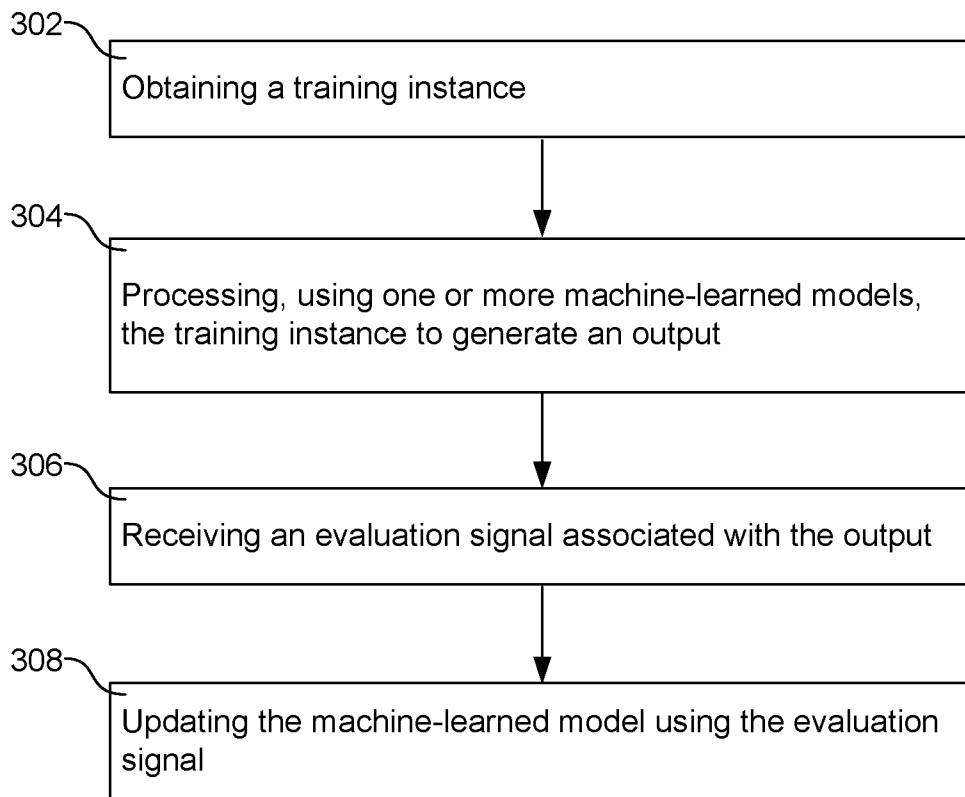


FIG. 2

3/12

300**FIG. 3**

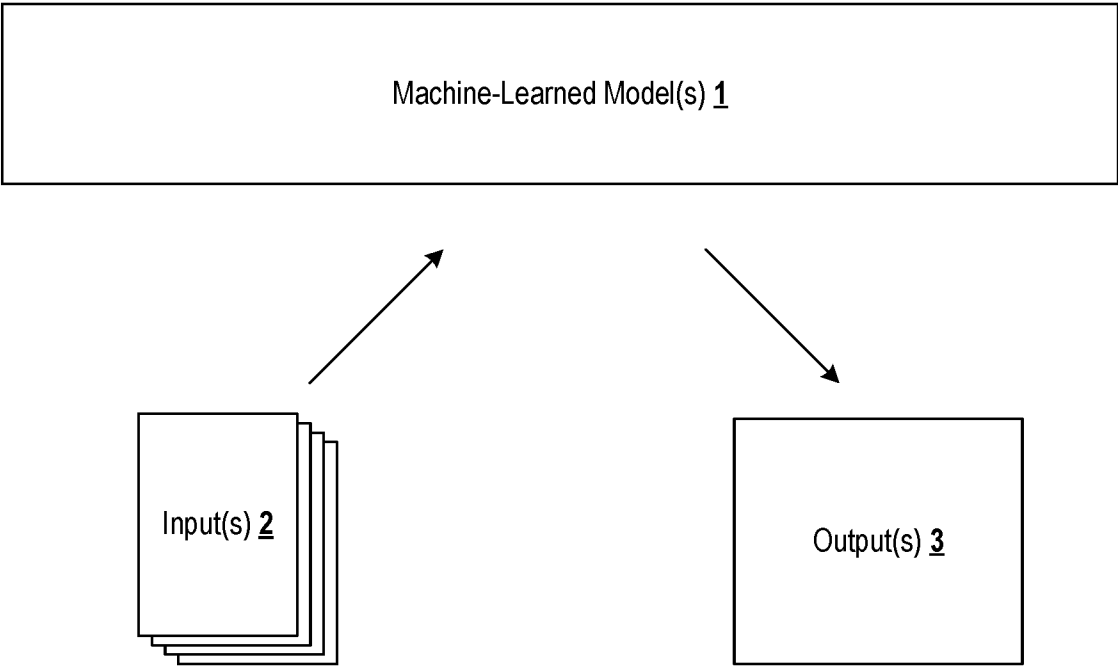


FIG. 4

5/12

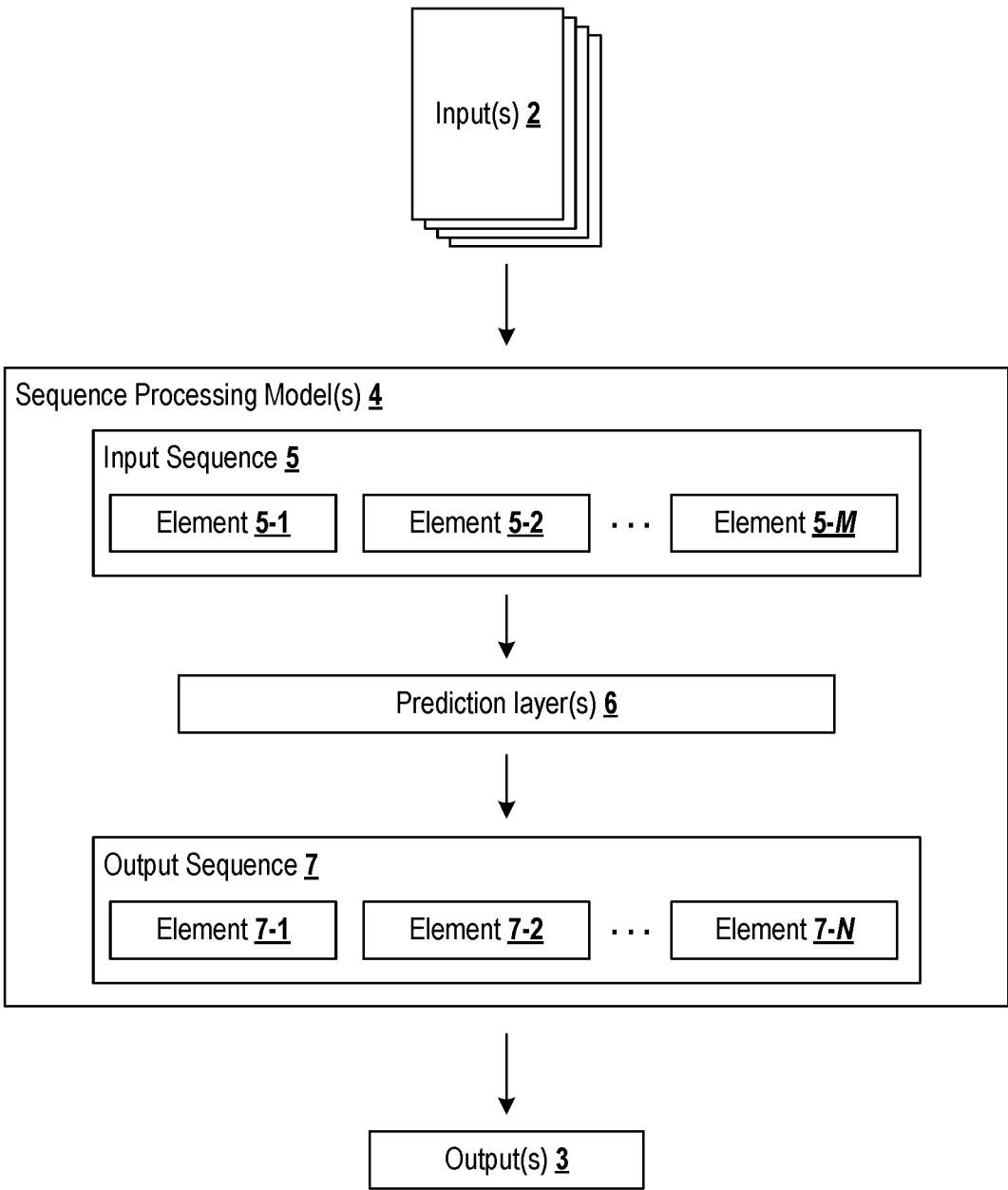


FIG. 5

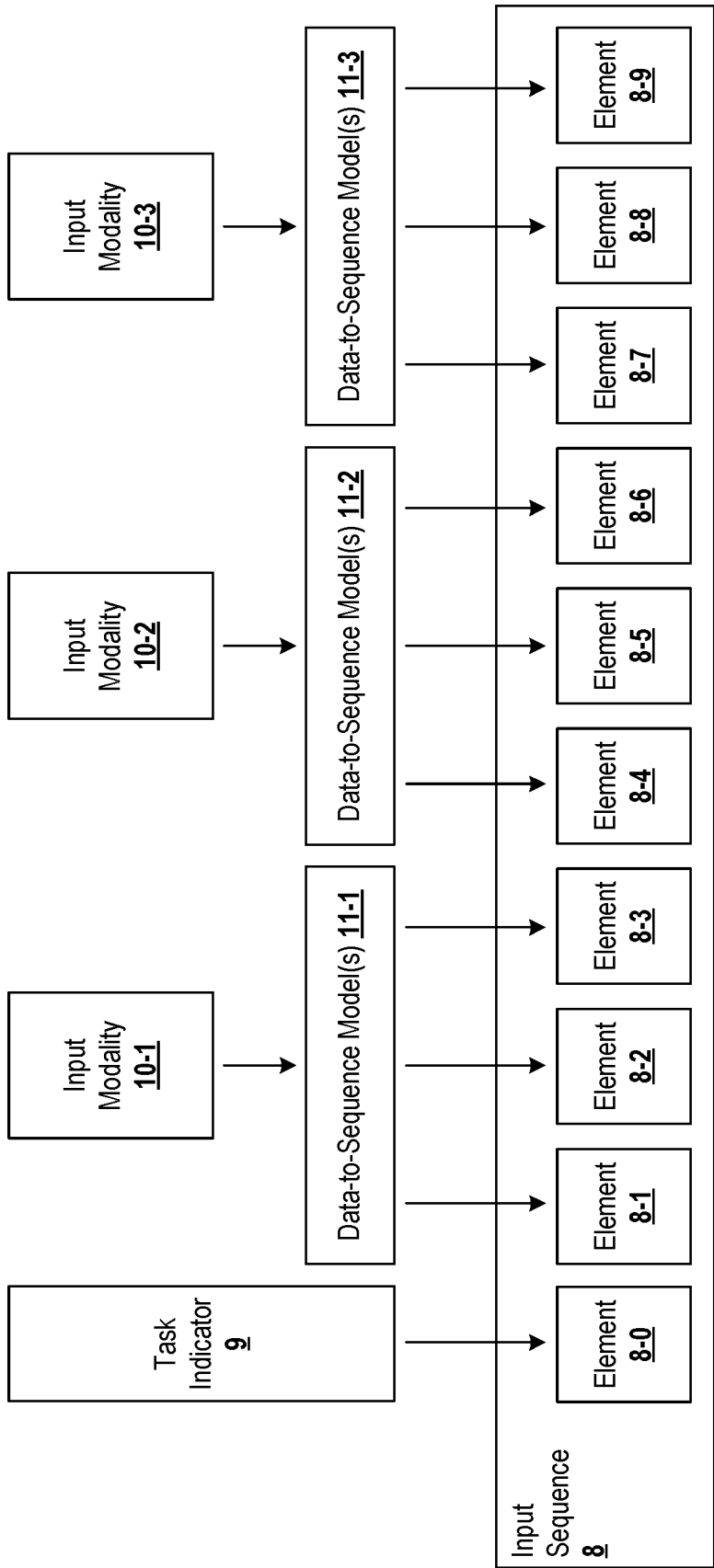


FIG. 6

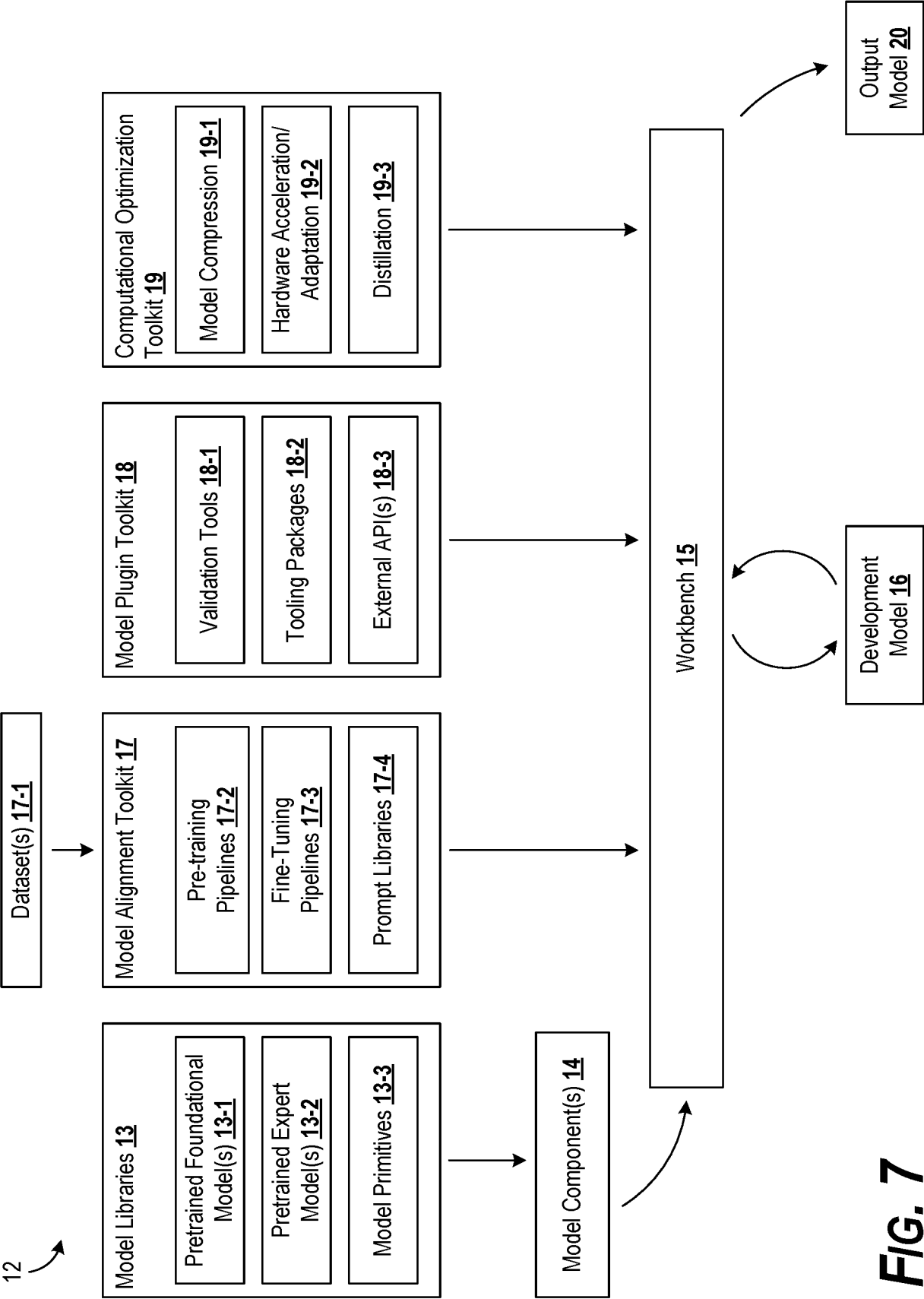


FIG. 7

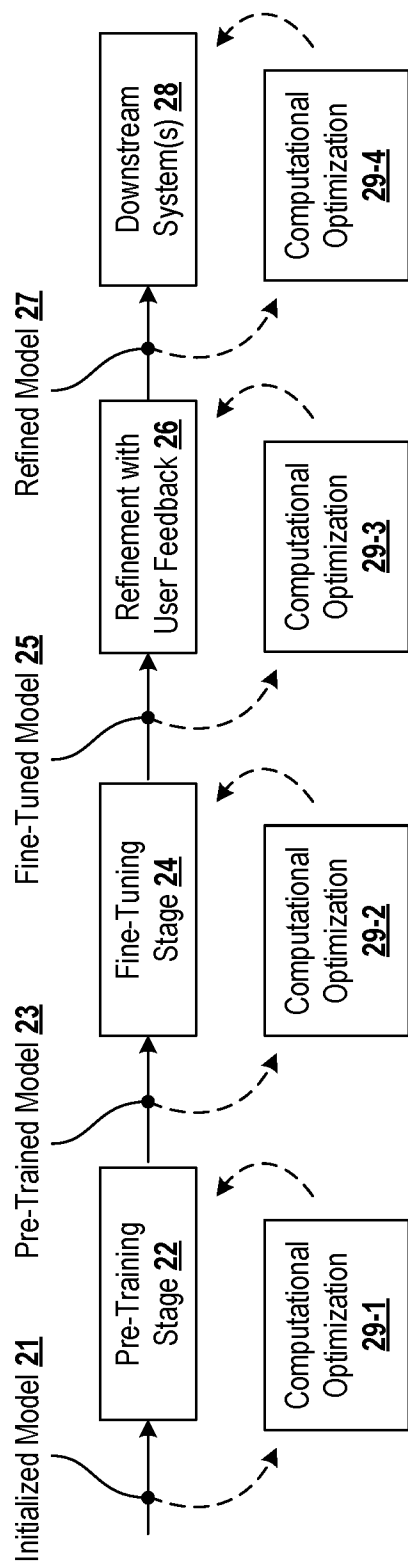


FIG. 8

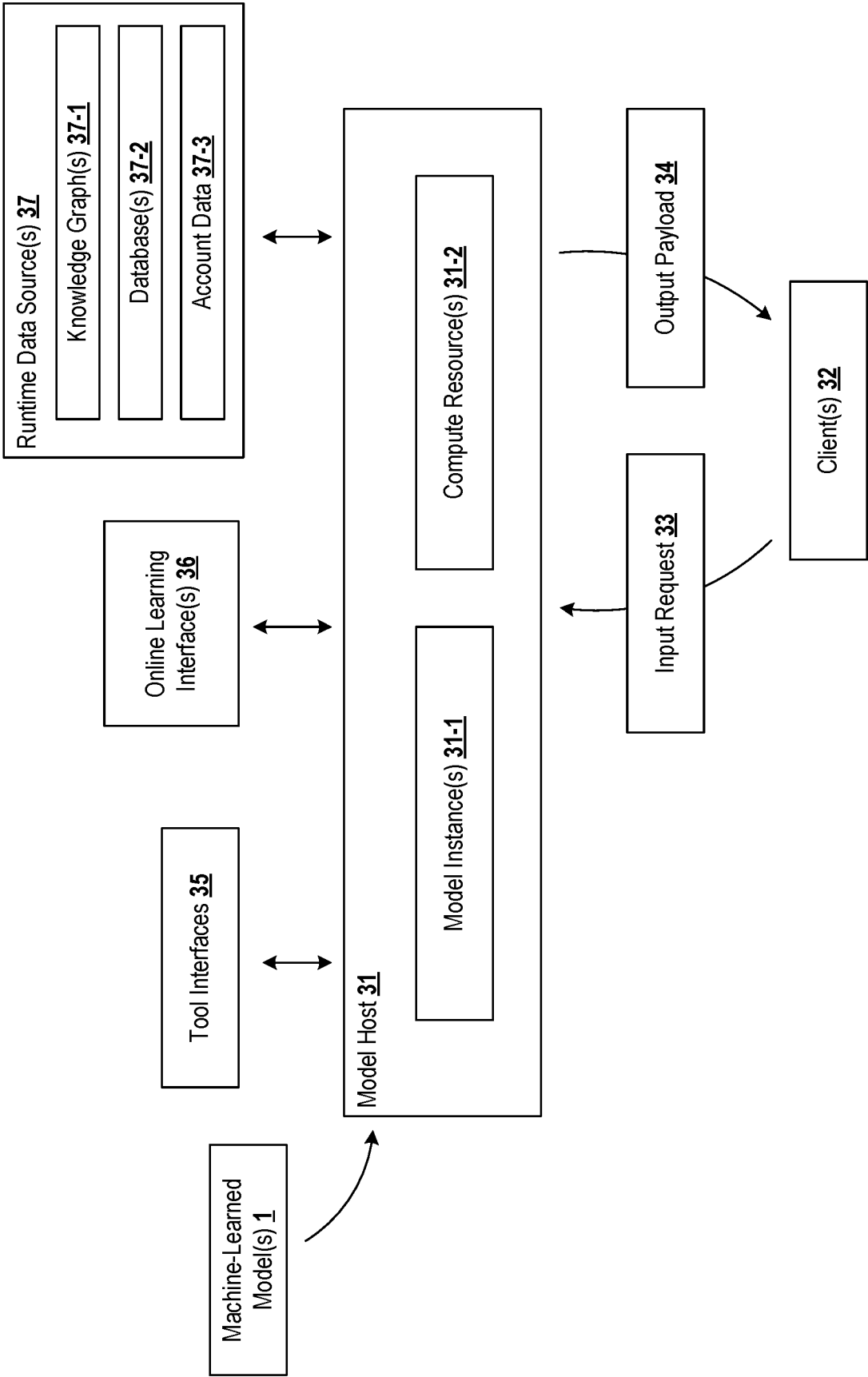


FIG. 9

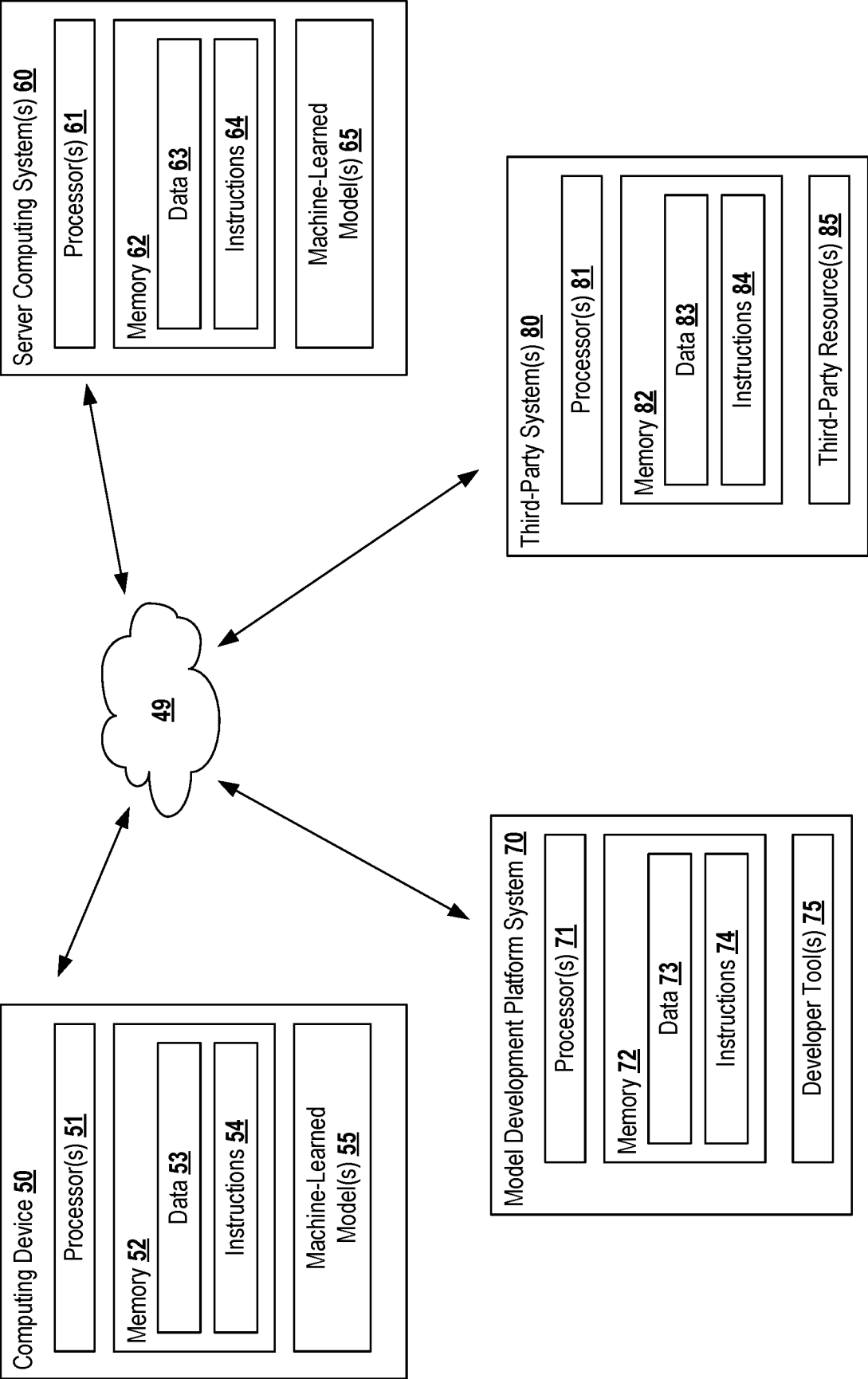


FIG. 10

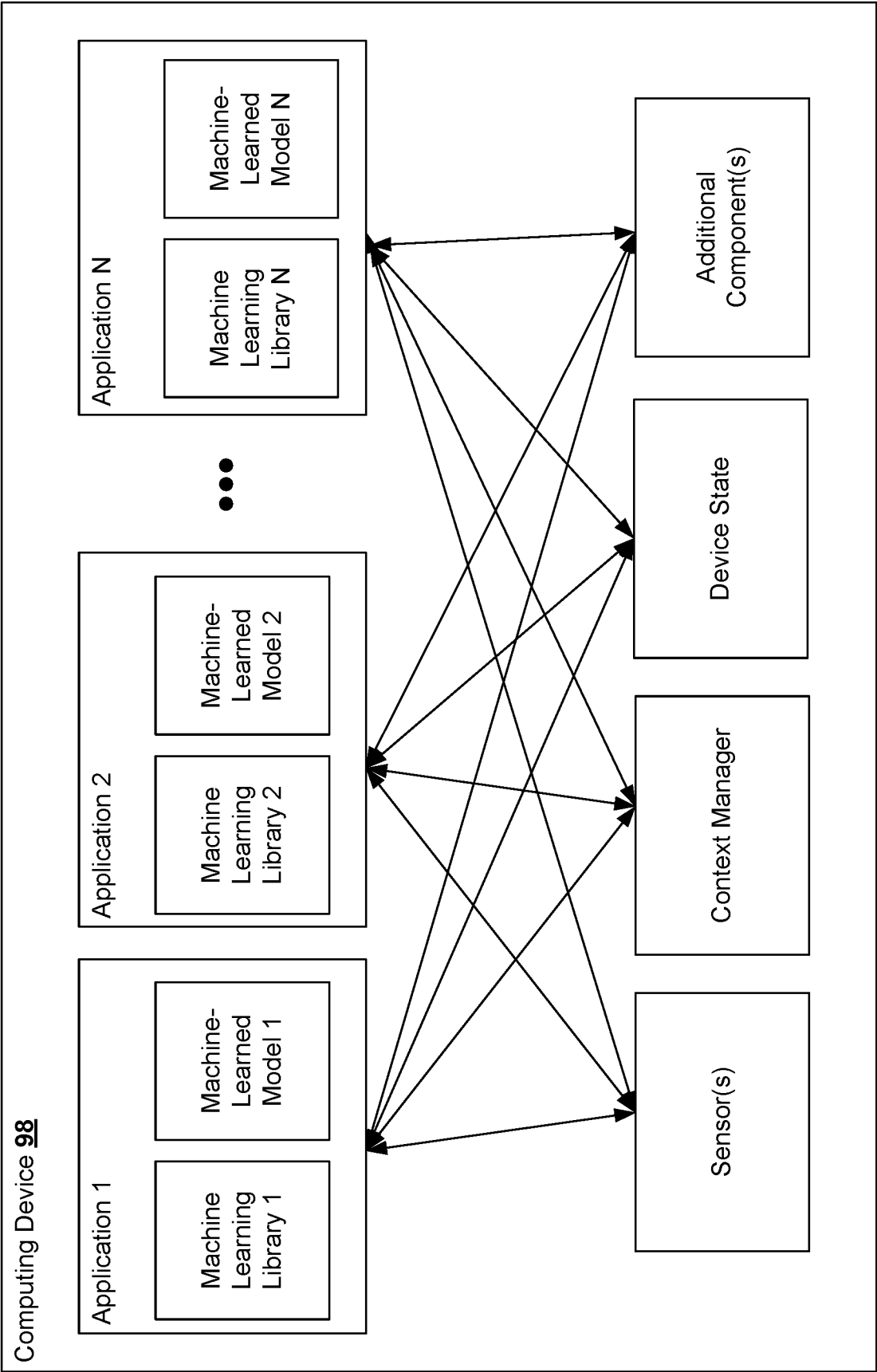


FIG. 11

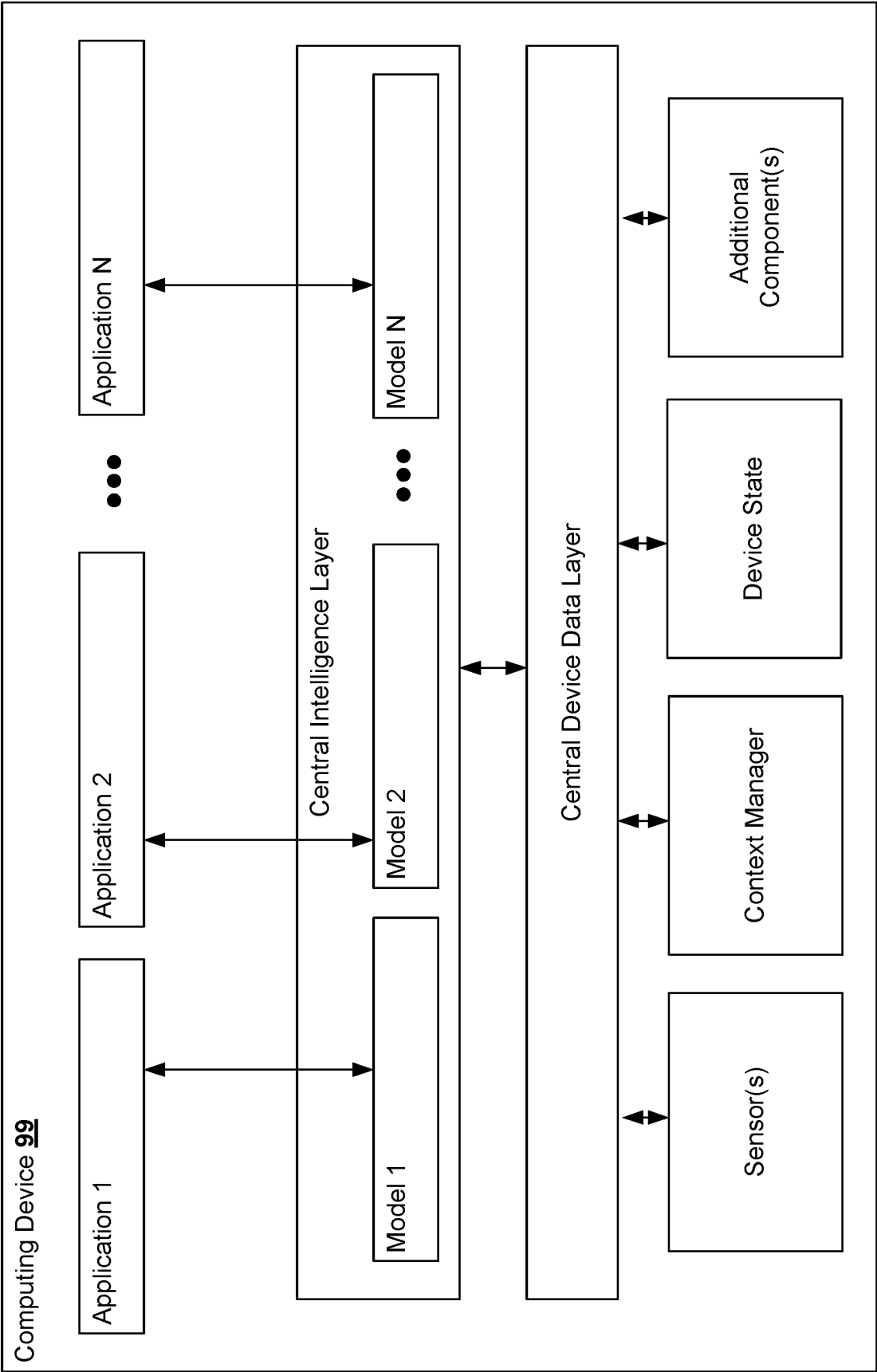


FIG. 12

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 24/22528

A. CLASSIFICATION OF SUBJECT MATTER

IPC - INV. G06F 16/33, G06F 40/284, G06F 40/40 (2024.01)

ADD. G06F 40/00 (2024.01)

CPC - INV. G06F 16/3344, G06F 40/284, G06F 40/40

ADD. G06F 40/00

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

See Search History document

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

See Search History document

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

See Search History document

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	US 2022/0093088 A1 (Apple Inc.) 24 March 2022 (24.03.2022), entire document, especially para [0026] - [0027], [0032] - [0033], [0210], [0236], [0238], [0243], [0245], [0248] - [0250], [0256], [0263], [0274], [0282], [0306], [0311], [0314]	1-6

☐ Further documents are listed in the continuation of Box C.☐ See patent family annex.

* Special categories of cited documents:

"A" document defining the general state of the art which is not considered to be of particular relevance

"D" document cited by the applicant in the international application

"E" earlier application or patent but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art

"&" document member of the same patent family

Date of the actual completion of the international search

31 May 2024 (31.05.2024)

Date of mailing of the international search report

JUL 25 2024

Name and mailing address of the ISA/US

Mail Stop PCT, Attn: ISA/US, Commissioner for Patents

P.O. Box 1450, Alexandria, Virginia 22313-1450

Facsimile No. 571-273-8300

Authorized officer

Kari Rodriguez

Telephone No. PCT Helpdesk: 571-272-4300

INTERNATIONAL SEARCH REPORT

International application No.

PCT/US 24/22528

Box No. II Observations where certain claims were found unsearchable (Continuation of item 2 of first sheet)

This international search report has not been established in respect of certain claims under Article 17(2)(a) for the following reasons:

1. ☐ Claims Nos.:
because they relate to subject matter not required to be searched by this Authority, namely:
2. ☐ Claims Nos.:
because they relate to parts of the international application that do not comply with the prescribed requirements to such an extent that no meaningful international search can be carried out, specifically:
3. ☒ Claims Nos.: 7-20
because they are dependent claims and are not drafted in accordance with the second and third sentences of Rule 6.4(a).

Box No. III Observations where unity of invention is lacking (Continuation of item 3 of first sheet)

This International Searching Authority found multiple inventions in this international application, as follows:

1. ☐ As all required additional search fees were timely paid by the applicant, this international search report covers all searchable claims.
2. ☐ As all searchable claims could be searched without effort justifying additional fees, this Authority did not invite payment of additional fees.
3. ☐ As only some of the required additional search fees were timely paid by the applicant, this international search report covers only those claims for which fees were paid, specifically claims Nos.:
4. ☐ No required additional search fees were timely paid by the applicant. Consequently, this international search report is restricted to the invention first mentioned in the claims; it is covered by claims Nos.:

Remark on Protest

- ☐ The additional search fees were accompanied by the applicant's protest and, where applicable, the payment of a protest fee.
- ☐ The additional search fees were accompanied by the applicant's protest but the applicable protest fee was not paid within the time limit specified in the invitation.
- ☐ No protest accompanied the payment of additional search fees.