

SYSTEMS AND METHODS FOR HYBRID ARTIFICIAL INTELLIGENCE ENHANCEMENT AND OPTIMIZATION

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This is a PCT patent application that claims benefit to U.S. provisional application serial number 63/539,102 filed on September 19, 2023 which is incorporated by reference in its entirety.

FIELD

[0002] The present disclosure generally relates to artificial intelligence (AI) including large language models (LLMs); and in particular to systems and methods for enhancing the capabilities of conversational AI systems to provide more accurate and reliable responses in enterprise settings across diverse industries.

BACKGROUND

[0003] Large language models such as ChatGPT can sometimes produce erroneous or hallucinated responses unsuitable for enterprise applications. These models generate answers probabilistically, providing human-like insights through immense statistical computation. However, the probability-based nature of these models leads to insufficient accuracy with respect to output. Furthermore, lacking mechanisms for preserving truth, these models can fabricate false information, and enhancing the models themselves often fails to resolve such hallucinations.

[0004] It is with these observations in mind, among others, that various aspects of the present disclosure were conceived and developed.

SUMMARY

[0005] The present disclosure provides a number of examples that describe systems and methods for enhancing the capabilities of conversational AI systems such as ChatGPT to provide more accurate and reliable responses in enterprise settings across diverse industries. In the context of the disclosed methods,

devices, techniques, apparatus, systems, and so on, the terms “operable to,” “configured to,” and “capable of” used herein are interchangeable.

[0006] In general, the innovation described herein aims to enhance the capabilities of conversational AI systems like ChatGPT to provide more accurate and reliable responses in enterprise settings across diverse industries including:

- Energy – Leverage utility grid data to provide outage diagnostics and restoration strategies. Plan renewable expansion based on generation assets.
- Finance – Make personalized investment recommendations based on an individual’s portfolio. Detect fraudulent transactions using knowledge of normal patterns.
- Transportation – Suggest optimized logistics plans considering factors like weather, traffic, fleet availability. Track assets using geospatial databases.
- Manufacturing – Interface with IoT sensors for predictive maintenance. Schedule production optimally based on demand forecasts and machine availability.
- Retail – Recommend Interface with IoT sensors for predictive maintenance. Schedule production optimally based on demand forecasts and machine availability.

[0007] The innovation’s grounding in logical reasoning, mathematical optimization, and integration with enterprise data systems allows it to move beyond general conversational abilities. It can provide tailored, validated solutions connected to the specific operational realities of different businesses and industries.

[0008] In a first set of illustrative examples, the innovation can take the form of a method comprising the steps of (a) accessing at least one response to a query from a large language model; (b) parsing the at least one response into logic programming predicates defining a plurality of facts and actions associated with the at least one response; (c) conducting an accuracy (correctness) validation operation for the at least one response in view of a knowledge base for logic programming including: (c)(i) verifying and maintaining facts of the plurality of facts in the at least one response shown to be true via the knowledge base, and (c)(ii) removing any other facts of the plurality of facts from the at least one response defining negations shown to be true in

the knowledge base; and (d) generating an output evaluating and returning a validated version of the at least one response based on the accuracy validation operation.

[0009] The method can further include steps of determining the plurality of facts cannot be verified by the knowledge base; and conducting category validation, including: generating one or more prompts that altered from the query but are semantically equivalent to the query, for each of the one or more prompts, obtaining a respective response from the large language model, establishing a structural relationship between the generated prompts and the corresponding responses from the large language model, and detecting for any semantic inconsistencies using category theory, wherein an identification of a semantic inconsistency reflects indicates a lack of reliability associated with the at least one response.

[0010] In a second set of illustrative examples, the innovation can take the form of a system including a processor in communication with one or more computing devices implementing a large language model (LLM). The processor parses an LLM response from the large language model to logic programming predicates, validates the logic programming predicates and by extension the LLM response for accuracy, and conducts category validation as described herein

[0011] In a third set of illustrative examples, the innovation can take the form of computer-executable instructions stored in a non-transitory medium (memory), the instructions executable by a processor to perform one or more of the operations of the method implementation, such as to parse the large language model response to logic programming predicates, validate the logic programming predicates and by extension the LLM response for accuracy, and to conduct category validation as described herein.

[0012] The foregoing examples broadly outline various aspects, features, and technical advantages of examples according to the disclosure in order that the detailed description that follows may be better understood. It is further appreciated that the above operations described in the context of the illustrative example method, device, and computer-readable medium are not required and that one or more operations may be excluded and/or other additional operations discussed herein may be included. Additional features and advantages will be described hereinafter. The

conception and specific examples illustrated and described herein may be readily utilized as a basis for modifying or designing other structures for carrying out the same purposes of the present disclosure. Such equivalent constructions do not depart from the spirit and scope of the appended claims.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] FIG. 1 is a general system diagram associated with the inventive concept described herein for enhancing the capabilities of conversational AI systems to provide more accurate and reliable responses.

[0014] FIG. 2A is an illustration of an example process and exemplary logic for conducting accuracy validation for a response elicited by an LLM via a query.

[0015] FIG. 2B is an illustration of an example process and exemplary logic for conducting category validation for the response of FIG. 2A.

[0016] FIG. 3 is an illustration of additional example architecture aspects of the system of FIG. 1 and the operations of FIGS. 2A-2B.

[0017] FIG. 4 is an illustration of example data flow aspects associated with the system of FIG. 1 and the operations of FIGS. 2A-2B.

[0018] FIG. 5 is an illustration of example aspects associated with multi-agent solver operations for supplementing the LLM response described herein.

[0019] FIG. 6 is an illustration of a IEEE node test feeder as described herein.

[0020] FIG. 7 is a simplified block diagram of an example computing device that can be implemented for the inventive concept described herein.

[0021] Corresponding reference characters indicate corresponding elements among the view of the drawings. The headings used in the figures do not limit the scope of the claims.

DETAILED DESCRIPTION

[0022] The inventive concepts described herein relate to examples of systems and methods for enhancing the capabilities of conversational AI systems to provide more accurate and reliable responses. In particular, the present innovation uniquely integrates logic programming, which represents truth and falsehood, with large language models. Operations defining a logic engine filters out incorrect responses using an enterprise knowledge base encoded in logic programming. The inventive concept can also include logic for generating optimized solutions matching user queries by combining model outputs with enterprise knowledge. An example software architecture is denoted GPTProX, and the model parses model responses into logic programming predicates. Note that Horn clauses can represent first-order logic; therefore, LLM responses can be parsed into predicates that serve as terms within Horn clauses. The logic programming predicates are validated against the knowledge base to eliminate hallucinations. By fusion of AI with mathematical optimization, the innovation generates precise, customized responses grounded in reality. This principled approach promises to unlock enterprise applications for large language models by overcoming inaccuracies through structured knowledge representation and reasoning.

[0023] In some examples, GPTProX includes a new software system that combines chatbots like ChatGPT with logic programming and math optimization tools. It does this to give more accurate and reliable answers for business uses. GPTProX can take ChatGPT's responses and check them using logic programming (e.g., in Prolog). This helps filter out wrong or made-up answers from ChatGPT. A validator based on category theory can also provide a backup check for accuracy.

[0024] GPTProX can connect to different databases such as Wikipedia, financial databases, and mapping databases. These databases can be leveraged to give better answers based on real data. When asked for helpful suggestions, GPTProX can use templates for linear programming, dynamic programming, and optimal control, and can run these templates in Matlab to give optimized solutions.

[0025] ChatGPT can then rephrase solutions from traditional LLMs to make them easier to understand. GPTProX can also make fully automated choices

using ChatGPT plus validated facts and multi-agent math solvers. This allows real-time, automated actions using both conversation and structured business data.

[0026] Overall, examples of GPTProX can combine chatbots, logical reasoning, math optimization, and databases to give customizable, accurate technical solutions for companies to overcome the challenges dealing with accuracy of presently available LLMs and similar platforms.

[0027] Introduction & Technical Problems

[0028] Business IT systems today, even those equipped with AI models, are limited with respect to performance and the types of tasks they can perform. While LLMs such as ChatGPT and other chatbots are not so limited in this regard, they can produce wrong or imagined answers and lack real-world business knowledge. In general, LLMs suffer from known technical issues associated with accuracy of responses elicited from various queries.

[0029] Example Technical Solutions ("GPTProX")

[0030] Examples of the inventive concept, sometimes denoted herein as "GPTProX," solve the aforementioned technical problems by combining Chat-GPT with real company systems. Logic programming can be implemented and configured to check ChatGPT's answers against knowledge bases. This improves the answers to fit the company's context.

[0031] GPTProX can include implementation of a dynamic knowledge base in Prolog that connects to one or more databases, GIS systems, niche data sources, and Wikipedia depending upon the given end-user, goals, and application. GPTProX can automatically expands its knowledge base using inductive logic programming. When answers are still uncertain, GPTProX can use category theory to catch inconsistencies that could mean made-up information.

[0032] GPTProX can divided answers into facts and actions. For actions, it can use templates for linear programming, dynamic programming, and optimal control to formulate the problem mathematically. It can then call the solvers to find the best solutions.

[0033] ChatGPT can further rephrase the optimized solutions into clear language. This lets GPTProX give users accurate, tailored answers that fit their business realities.

[0034] So in summary, GPTProX can combine chat-bots, logic, math optimization, and live data to overcome limitations and inaccuracies. This levels up ChatGPT for real enterprise examples and can significantly improve accuracy of output derived from LLM models.

[0035] EXAMPLE IMPLEMENTATIONS

[0036] FIGS. 1-5 show example implementations including operations and components for enhancing the capabilities of conversational AI systems to provide more accurate and reliable responses. As indicated, an inventive concept responsive to the aforementioned technical problems and challenges can take the form of a computer-implemented system, designated system 100, comprising any number of computing devices or processing elements. In general, the system 100 leverages logic programming with large language models (LLMs) to validate and enhance responses to queries made to the LLMs. While the present inventive concept is described primarily as an implementation of the system, it should be appreciated that the inventive concept may also take the form of tangible, non-transitory, computer-readable media having instructions encoded thereon and executable by a processor, and any number of methods related to examples of the system described herein.

[0037] FIG. 1 illustrates example system components associated with the inventive concept described herein. As indicated in FIG. 1, the system 100 includes at least one processor 102, and at least one of a memory 103 or storage device 103 storing instructions 104 accessible by the processor 102 to perform various functions and operations described herein, and the system 100 validates and otherwise provides output associated with at least one response 132 generated by at least one large language model 130. The system 100 can further include a network interface 106 (or multiple network interfaces), and a bus (or wireless medium) for interconnecting and/or establishing communication between the aforementioned components. The network interface 106 includes the mechanical, electrical, and signaling circuitry for communicating data over links (e.g., wires or wireless links) within a network (e.g., the

Internet). The network interface 106 can be configured to transmit and/or receive data using a variety of different communication protocols, as will be understood by those skilled in the art.

[0038] In general, the processor 102 is configured (via the instructions 104) to perform operations including validation of the at least one response 132 generated by the least one large language model (LLM) 130 in response to at least one of a query 128 fed to or received by the LLM (by an end user or otherwise). The LLM 130 can be implemented via any number or type of computing elements including cloud implementations, and examples of the LLM 130 are described herein. In various examples, the processor 102 accesses input data 114A provided to an external computing device 108 (e.g., by engaging a user interface (UI) 112) rendered along a display 110), and the processor 102 can return an output (e.g., a modified and/or validated response to a query made to an LLM) in the form of output data 114B for access by the computing device 108. Input data 114A can include the response 132 or data associated with the response generated by the LLM 130, and other information needed to perform operations described herein. Output data 114B can include information about validation and enhancement associated with the response 132. For example, the output data 114B can include a modified response (e.g., some change to the original response 132 generated by the LLM 130), a confirmation that the response 132 is validated and/or verified, and/or supplemental information or functions for enhancing the response 132 as described herein.

[0039] In some examples, the processor 102 can access data from one or more data source devices 120 shown by example as device 120A, device 120B, and device 120C. Devices 120 can provide any datasets or information needed to perform LLM response validation and/or LLM response enhancement operations or any other operations described herein. Datasets and other information can be preprocessed and stored within a database 118 as shown. Any artificial intelligence (AI) models as referenced herein can include classification models, supervised or unsupervised learning models such as K Nearest Neighbors, linear regression models, neural networks, deep learning models, etc.

[0040] In general, the instructions 104 can be implemented as code and/or machine-executable instructions executable by the processor 102 that may represent one or more of a procedure, a function, a subprogram, a program, a routine, a subroutine, a module, an object, a software package, a class, or any combination of instructions, data structures, or program statements, and the like. In other words, the instructions 104 or any operations performed by the processor 102 described herein may be implemented by hardware, software, firmware, middleware, microcode, hardware description languages, or any combination thereof. When implemented in software, firmware, middleware or microcode, the program code or code segments to perform the necessary tasks (e.g., a computer-program product) may be stored in a computer-readable or machine-readable medium (e.g., the memory 103), and the processor 102 performs the tasks defined by the code.

[0041] FIG. 2A illustrates an example process 200 or method implementation associated with the system of FIG. 1, and FIGS. 3-5 illustrate further exemplary architecture and data flow aspects associated with the process 200 and system components introduced in FIG. 1. Referring to block 201 of FIG. 2, via the user interface 112 or otherwise, the processor 102 can access the response 132 and/or data defining the response 132 generated by the LLM 130. The processor 102 can also access the query 128 and/or data defining the query 128 provided to the LLM 130; the query 128 being the communication that results in the elicitation of the response 132 from the LLM 130.

[0042] In some examples, the processor 102 can be provided with or otherwise access the data defining the response 132 generated by the LLM 130 without communicating with the LLM 130. Alternatively, the processor 102 can receive the query 128 directly and/or can receive the response 132 directly from the LLM 130 and can otherwise facilitate communication between an end user and the LLM 130 to initiate output of the response 132 and modifications or validations thereof. In some examples, the query 128 can be defined as part of the input data 114A that can be sent to an Application Programming Interface (API) wrapper associated with the LLM 130 as indicated in FIG. 3 to elicit the response 132. Any such variations are contemplated, but

ultimately the processor 102 at least access the response 132 and the query 128 to perform the validation and optimization functions described herein.

[0043] In the particular (non-limiting) example shown in FIG. 3, the query 128 can be accessed or made at the user interface 112. The query 128 is any query, prompt, or other communication made to elicit the response 132 from the LLM 130. Example queries provided herein, but a one example of the query 128 could be as follows.

QUERY:

“Logician Hao Wang Affiliations”

This example type query could be made via a chatbot associated with the LLM 130, for example (e.g., fed to ChatGPT). While FIG. 2A indicates the query 128 originating via the user interface 112, in other examples the processor 102 can access the response 132 generated from the LLM 130 and the query 128 can be generated elsewhere.

[0044] Specific examples of the response 132 generated by the LLM 130 or output responsive to the query 128 are provided herein, but continuing the above example, one example of the response 132 associated with the specific example query above (“Logician Hao Wang Affiliations”) could include the following.

LLM RESPONSE:

Hao Wang was affiliated with several prestigious institutions over the course of his career.

Wang received his undergraduate degree from the National Southwestern Associated University in Kunming, China. He then went on to earn his Ph.D. from Harvard University in the United States, where he also held a Junior Fellowship from 1947 to 1950.

He was a faculty member at the University of Notre Dame from 1950 to 1955, then joined the Institute for Advanced Study in Princeton, where he worked closely with the famous mathematician Kurt Gödel.

In 1956, Wang accepted a position at the Rockefeller University in New York City, where he was a faculty member for the rest of his career. He held the title of Professor of Logic until his retirement.

It's also worth noting that he was a visiting professor at various institutions worldwide, including the University of Oxford.

While the example response 132 provided above is text-based, other iterations of the response 132 can include graphical images, numerical output and/or functions, or other output information as shown in other examples provided herein.

[0045] Referring to block 202 of FIG. 2, the processor 102 parses the response 132 into logic programming predicates defining a plurality of facts and actions/rules associated with response. In some examples, the processor 102 can implement a Prolog validation engine or functionality thereof defined by the instructions 104 to parse the response 132 in this manner to Prolog format. Continuing the above example, the above response can be parsed to generate the following logic programming predicates:

```

person(hao_wang).
position(hao_wang, harvard_university, phd_student, 1948).
position(hao_wang, harvard_university, junior_fellow, 1947).
position(hao_wang, harvard_university, assistant_professor, 1948).
position(hao_wang, harvard_university, professor_logic_and_applied_mathematics,
1961).

position(hao_wang, university_of_notre_dame, faculty_member, 1950).
position(hao_wang, institute_for_advanced_study, faculty_member, 1955).

position(hao_wang, university_of_oxford, reader_in_philosophy_of_mathematics,
1956).
position(hao_wang, rockefeller_university, faculty_member, 1956).
position(hao_wang, rockefeller_university, head_of_logic_research_group,
1967).
position(hao_wang, rockefeller_university, professor_of_logic, 1967).

```

[0046] As shown above, parsing of the response 132 to logic programming predicates as described transforms the response 132 into output defining logical programming sentences associated with the response 132 including conjunctions that structure relationships between the logical programming sentences.

[0047] Referring to block 203 of FIG. 2, the processor 102 can perform an accuracy validation operation for the response 132 in view of a knowledge base or otherwise to determine whether the response 132 is valid and/or should be modified in some form. In some examples, the facts defined by the response 132 as parsed can be validated with a Prolog Knowledge base (illustrated in FIG. 3, and developed according to the examples described herein). In some examples, accuracy validation includes (i) verifying and maintaining facts of the plurality of facts in the response 132 shown to be true via a knowledge base, and (ii) removing any other facts of the plurality of facts from the logic programming predicates response 132 defining negations shown to be true in the knowledge base. More specifically, in some examples, if the negation of a predicate of the logic programming predicates is determined by the processor 102 to be True in view of the knowledge base, the processor 102 executes logic to remove this predicate from the logic programming predicates associated with the response 132. If the predicate is determined to be True in knowledge base, then the processor 102 can execute logic to accept the predicate of the logic programming predicates. If facts of the logic programming predicates and its negation are deemed not true in the knowledge base, the processor 102 can implement category validation as further described herein. In addition, for action predicates of the logic programming predicates, the processor 102 can implement a multi-agent optimization solver for further functionality as shown in FIG. 5 and further described herein.

[0048] Continuing with the example above example associated with the example of the query 128 ("Logician Hao Wang Affiliations"), the processor 102 conducting the accuracy validation in view of the knowledge base as described determines the following.

```
position(hao_wang, university_of_notre_dame, faculty_member, 1950)  
is False.
```

Specifically, as shown, the above proposition defined by the shown predicate and associated facts from the example of the response 132, asserting that Hao Wang is from or affiliated with the University of Notre Dame and was a faculty member in 1950,

is determined to be false by the processor 102 referencing the subject predicate in view of the knowledge base. In some examples, this determination by the processor 102 includes executing a “Consult” operation to a Prolog file of the knowledge base, where the knowledge base is created at least in part using Wikipedia or some other data source.

[0049] Referring to block 204 of FIG. 2, the processor 102 generates an output evaluating and returning a validated version of the at least one response based on the accuracy validation operation. In some examples, where all of the plurality of facts are determined to be true and valid by reference to the knowledge base, the output can include a simple validation confirmation. However in other examples, such as the example above dealing with the query 128 “Logician Hao Wang Affiliations” and the associated response 132, the output can include a *modified* response or modified version of the response 132. For example, a modified version of the response 132, presented back to an end user device or otherwise, can take the following form or something similar:

Hao Wang was affiliated with several prestigious institutions over the course of his career.

Wang received his undergraduate degree from the National Southwestern Associated University in Kunming, China. He then went on to earn his Ph.D. from Harvard University in the United States, where he also held a Junior Fellowship from 1947 to 1950.

He was a faculty member at the University of Notre Dame from 1950 to 1955, then joined the Institute for Advanced Study in Princeton, where he worked closely with the famous mathematician Kurt Gödel.

In 1956, Wang accepted a position at the Rockefeller University in New York City, where he was a faculty member for the rest of his career. He held the title of Professor of Logic until his retirement.

It's also worth noting that he was a visiting professor at various institutions worldwide, including the University of Oxford.

Note that the language indicated above in strikethrough can be omitted in the modified response because such information has been determined by the processor 102 to be

erroneous or unverifiable in view of the knowledge base (by comparison of the associated predicate to the data of the knowledge base).

[0050] Referring to FIG. 2B, an example process 250 is illustrated providing further validation operations that can be implemented by the processor 102 related to the operations shown in FIG. 2A. In block 251 of process 250, the processor 102 first determines that the plurality of facts, extracting from the parsing operations of FIG. 2A, cannot be verified or validated by the knowledge base for whatever reason.

[0051] Referring to blocks 252-255, the processor 102 can conduct category validation to catch inconsistencies associated with the response 132 that could indicate the response 132 is constructed at least in part by made-up information. In other words, the subject category theory component vets responses from the LLM 130 by modeling them as categorical objects mapped from input prompts/queries. Deviations in the mappings allow for detecting inconsistencies.

[0052] In some examples, category validation operations executable by the processor 102 include generating one or more prompts that are altered from the query 128 but are semantically equivalent to the query 128, for each of the one or more prompts, obtaining a respective response from the large language model 130, establishing a structural relationship between the generated prompts and the corresponding responses from the large language model 130, and detecting for any semantic inconsistencies using category theory, wherein an identification of a semantic inconsistency reflects indicates a lack of reliability associated with the at least one response.

[0053] FIGS. 3-5 illustrate further non-limiting steps (1-35) that can be implemented for the operations of FIGS. 2A-2B, or otherwise, as follows:

1. ① User interface. The processor 102 can take user input in as text in English or spoken in English. The input can be translated or assembled in English when the user finishes the input. Then the input can be sent from the processor 102 to the (Large Language Model) LLM 130, which can include a API wrapper.
2. ② A Prolog Validation Engine can be implemented to conduct the Parsing of a response from an LLM such as a ChatGPT response to a Prolog format

(defining logic programming predicates). The step demonstrates validating the correctness using Prolog knowledge base. There are two different type of prolog statements in a response. Facts and actions. For facts, validate with Prolog Knowledge base. If the negation of a predicate is true in knowledge base, remove this predicate. If the predicate is True in knowledge base, accept the predicate. For facts and its negation are not true in knowledge base, send to Category validation engine for further process. For action predicates, send to multi-agent optimization solver for detailed action.

3. ③ A Category Validation Engine can be used to Category Filter and detect a possible Hallucination. If detected, this step includes notifying the Prolog Validation Engine for query re-alignment.
4. ④ Multi-Agent Optimization Solver: A Prolog Knowledge Base can be implemented to store and reference domain specific subject matter and respective problem formulation templates can further be maintained to support automated solution generation (e.g., multi-agent solver for supplementing response 132). Available solvers offered by Matlab toolbox can be installed. An underlying problem can be decomposed to modal problems that can be solved by an Agent. Multiple Agent joint solutions can be achieved by sharing the constraints of each of the Agent solutions. Each agent solution can be a solution of an optimization problem.
5. ⑤ Prolog Knowledge Integrator, Two parts: 1. Prolog DB built by instruction-level parallelism (ILP)P. 2. Dynamically query Wikipedia and other Enterprise DBs
6. ⑥ Interface to Wikipedia: A compose html operation can be implemented to get a request to Wikipedia for the underlying subject. The response can be parsed into Prolog format by a parser. Next the step can include operations to update the Prolog knowledge base with the parsed Wikipedia response.
7. ⑦ Interfaces to enterprise business databases: query enterprise database for facts. For example, retrieve the name of CEO.
8. ⑧ Interfaces to enterprise utility GIS: manage geo-queries for location and geometric properties of an asset. Parse the result into Prolog format to be loaded to Prolog Knowledge Base.
9. ⑨ Interface to enterprise Ontology Database: Query the ontology database for the domain specific definition of the underlying terms and their equivalence. Parse the result to Prolog format and accept the file as Prolog facts and rules.

10. ⑩ LLM (Large Language Model) wrapper. Software package that keep the same interface with GPTProX. It can be customized to all available LLM's.
11. ⑳ Prolog Based Domain Specific Engine. It takes care of two things: facts validation and action completion.
12. ㉑ Multi-Agent optimization solver.
13. ㉒ LLM wrapper.
14. ㉓ User Input service.
15. ㉔ User initial prompt.
16. ㉕ Fetch user prompt to ㉒.
17. ㉖ LLM response to ㉑.
18. ㉗ Adjusted prompt based on the process of ㉑ to send to ㉒ for iteration.
19. ㉘ LLM response to ㉒ in iterations.
20. ㉙ Final response to user.

Relationship- ① and ⑩: ① Takes user input to make it ready as LLM (Large Language Model) query. ① send the query to ⑩. The message travels from ① to ⑩ one direction.

Relationship- ② and ③: ② send a pair of query and response to ③ for category based explorations. ③ send to ② "pass" or "fail" the category validation.

Relationship- ② and ④: ② send the attributes and specification of the solutions to achieve. ④ send the solutions of the underlying subject to ②.

Relationship- ② and ⑤: ② send the facts to ⑤ for validation and for additional information about the facts. ⑤ send true or false conclusions to the facts. If it is true, additional information to the best of the system's knowledge will be sent back to ② as well.

Relationship- ⑤ and ⑥: ⑤ send query to ⑥ for fact retrieval. ⑤ parses the response to Prolog format and load to Prolog Knowledge Base as True facts.

Relationship- ⑤ and ⑦: ⑤ send query to ⑦ for facts and rules of the enterprise. ⑤ parses the response to Prolog format and load to Prolog as facts and rules.

Relationship- ⑤ and ⑧: ⑤ send geospatial queries to ⑧ for asset location, asset connectivity, best route etc. ⑧ parses the response to Prolog format and load the result to Prolog Knowledge Base.

Relationship- ⑤ and ⑨: ⑤ send query to ⑨ for domain specific definition of terms and its equivalent terms. ⑤ parses the result to Prolog format to load Prolog Knowledge Base.

Relationship- ① and ⑩: ① send language prompt (query) to ⑩

Relationship- ② and ⑩: ② send language prompt (query) to ⑩. ② receive prompt response (queried by ① or ②) from ⑩.

Relationship- ③ and ⑩: ③ send language prompts and receive responses to ① for Category validation run.

In ②: If a response message are approved by Prolog Knowledge Base for facts and coordinate with ④ for action, then send the message to ①

If a predicate parsed from response message needs validation and the Prolog Knowledge Base can't approve or disprove, then send the predicate to ⑤ for validation or fact retrieval.

If an action predicate is an action, then send it to ④ for optimization.

If a fact predicate can't be approved or disproved by Prolog knowledge base and ⑤, then send it to ③ for category validation.

In ③: If the Category validation iteration (between ③ and ④) is not completed yet, then continue the iteration between ③ and ④.

If the iteration is done, then send the result to (2).

In ⑤: If a set of predicates are about a person or organization, then send the query to ⑥ for completion.

If a predicate is for a person in this enterprise, then send a query to ⑦ for knowledge completion.

If a predicate is about an asset with location, then send a query to ⑧ for completion.

If a predicate is an unknown concept, then send a query to ⑨ for its definition and its equivalent names.

In ③: If it is in the Category validation loop, then the messages are sent to 10. If the validation is done, then send the result to ②.

[0054] Constructing a ProLog Knowledge Base

[0055] The following non-limiting/example approach can be used for initial construction and continue enhancement to the knowledge base previously described. The knowledge base can incorporate both manually encoded and automatically extracted knowledge via web scraping, APIs, etc.

[0056] Domain experts codify key regulations, policies, and principles into logic programming rules in Prolog. This establishes a foundational set of validated facts spanning critical governing constraints.

[0057] Inductive logic programming techniques are then leveraged to learn new rules and facts from structured enterprise data sources. The system actively interfaces with databases like geographic information systems and financial systems to extract knowledge through inductive learning. Interface code enables two-way communication, supporting robust fact checking.

[0058] Additionally, Monte Carlo simulations model the behavior of ChatGPT's responses for the target domain. By analyzing response patterns and characteristics over many simulation iterations, the system derives supplemental rules to further enhance the Prolog knowledge base.

[0059] This hybrid approach combining manual encoding, inductive machine learning, and simulation-based discovery allows efficiently developing a comprehensive validated knowledge base. The continuous integration with live

enterprise data and logic extracted from conversational model observations enable the knowledge to stay relevant and complete as the application environment evolves.

[0060] Building an English to Prolog Parser with SpaCy and Ontology

[0061] Building an English text to Prolog parser can be done using SpaCy, an open source natural language processing library, together with an ontology database. This combination can help address ambiguities and domain-specific nuances. The general steps include:

- Develop the ontology database with structured knowledge about the domain, including concepts, relationships, and properties.
- Use SpaCy for text parsing into tokens, named entities, part-of-speech tags, etc. Customize it to recognize domain entities based on the ontology.
- Convert the parsed text into Prolog predicates based on the ontology structure and target Prolog knowledge base format.
- Handle negation, modifiers, and other complex linguistic constructs when translating to Prolog.
- Iteratively improve the parser by analyzing ChatGPT interactions and expanding the ontology.
- Implement a user feedback loop to refine the parser and knowledge base over time.

[0062] The combination of an ontology and SpaCy provides a good foundation. Focused domain knowledge and continuous improvement will allow the system to parse English into target Prolog representations with increasing accuracy.

[0063] Building a Fact-Checking System for ChatGPT Using NLP and Logic Programming

[0064] To develop a system that cross-references ChatGPT's answers with a reliable knowledge source, the following procedure can be implemented:

1. Text Parsing with SpaCy:

- See details in subsection “Building an English to Prolog Parser with SpaCy and Ontology”.

2. Conversion to Prolog Structure:

- Convert the parsed data into Prolog’s format of facts and rules.
- Transform subject-verb-object connections into Prolog predicates with designated arguments.
- Convert recognized named entities into Prolog’s constants.

3. Constructing a Knowledge Base in Prolog:

- Draft rules encapsulating ontological insights.
- Construct a Prolog database with predetermined facts and regulations that signify trusted information.
- Incorporate authoritative sources to establish facts about entities and their interconnections.
- Collaborate with specialists to formulate domain-specific regulations and principles.
- Regularly augment this database for comprehensiveness.

4. Verification of ChatGPT’s Assertions:

- Probe the Prolog database to authenticate ChatGPT’s statements.
- Design a query for each assertion to ascertain its validity.
- If a claim is contradicted, classify it as incorrect.

5. Enhancement Measures:

The prompt is: The branches of ieee13 circuit.

[0065] Local Knowledge Base Contents

[0066] Here is a Prolog program to describe the connection between

nodes:

```
% branch_factes
branch_fact(650, 632).
branch_fact(632, 633).
branch_fact(632, 645).
% Rule
branch(X, Y) :- branch_fact(Y, X).
branch(X, Y) :- branch_fact(X, Y).
```

[0067] Steps include:

- Store frequently used queries and their outcomes to expedite the process.
- Where feasible, evaluate multiple assertions collectively.
- Ensure robust error management during the stages of parsing, translating, and querying.
- Catalog and scrutinize scenarios causing discrepancies.

6. textbfContinuous System Refinement:

- Bolster each component to increase system resilience.
- Engage domain experts for assessments.
- Allow specialists to corroborate the parsed claim representations.
- Capitalize on expert evaluations to refine the parser and knowledge repository.

[0068] By adhering to this guide, one can craft a resilient fact-verification system for ChatGPT underpinned by NLP and logical programming techniques. The essence lies in metamorphosing linguistic content into logical entities that facilitate effective cross-referencing with authoritative data.

[0069] Parsing and validation example: IEEE13 Test Feeder

[0070] Referring to FIG. 6, to demonstrate how to integrate an LLM such as ChatGPT with an engineering application, one can use IEEE 13 Node Test Feeder circuit, originally published by the IEEE Distribution System Analysis Subcommittee Report.

```

branch_fact(645, 646).
branch_fact(633, 634).
branch_fact(632, 671).
branch_fact(671, 680).
branch_fact(671, 692).
branch_fact(671, 684).
branch_fact(684, 611).
branch_fact(684, 652).
branch_fact(692, 675).
% Rule
branch (X, Y): - branch_fact(Y, X).
branch (X, Y): - branch_fact(X, Y).

```

[0071] ChatGPT response to IEEE13 Test Feeder

[0072] ChatGPT prompt: IEEE13 test feeder, including connectivity using

branch ChatGPT response:

The IEEE 13 node test feeder is a highly unbalanced network that is often used for testing new power system algorithms. Its topology is that of a typical urban residential area, and it's known for its simplicity and yet enough complexity to test the various aspects of power system studies.

The topology of the IEEE 13 Node Test Feeder is as follows (shown using branches between the nodes):

```

branch(650, 632). branch(650, 671).

branch(632, 633). branch(632, 645).

branch(645, 646).

branch(633, 634).
branch(671, 680).    branch(671, 684).
branch(671, 692).

branch(684, 611). branch(684, 652).
branch(692, 675). branch(692, 751).

```

This list of branches represents the distribution lines connecting the nodes in the IEEE 13 Node Test Feeder. The topology of the system is that of a typical urban residential area, with some nodes having more than one distribution line connected to them.

[0073] “Consult” to the Prolog file that was created using diagram of IEEE13 nodes directly, the propositions

```
branch(650, 671) .
branch(692, 751) .

are False.
```

[0074] Category Theory Based Hallucination Detection

[0075] 1. Categories

- (a) P (Prompts)
 - i. Objects: ChatGPT prompts.
 - ii. Morphisms: Functions transforming one prompt into another while preserving semantics.
- (b) R (Responses)
 - i. Objects: ChatGPT responses.
 - ii. Morphisms: Functions transforming one response into another, like paraphrasing or refining, without losing the intended meaning.

[0076] 2. Functor F

- (a) Maps the transformation of prompts in category P to the resultant transformations in responses in category R.
 - i. Object Mapping: For every prompt A in P, $F(A)$ in R is the ChatGPT response to A .
 - ii. Morphism Mapping: For a transformation $f: A \rightarrow B$ in P, $F(f): F(A) \rightarrow F(B)$ in R represents a correlated transformation between the responses.

[0077] 3. Monadic Structure on R

- (a) To encapsulate "computational effects" on responses:

- i. Endofunctor T : Represents computational processes on responses, such as refinement or expansion.
- ii. Natural Transformations:
 - A. Unit (η): Embeds a response into the monadic context of T .
 - B. Multiplication (μ): Combines or simplifies two consecutive applications of the computational process T .

[0078] 4. Operational Process

- Semantic Prompt Generation: Use morphisms in P to generate various semantically equivalent prompts.
- (b) Response Collection: For each generated prompt, obtain a ChatGPT response.
 - (c) Mapping via Functor F : Establish the structural relationship between the generated prompts and the corresponding responses.
 - (d) Monadic Processing: Introduce and manage the computational effects on the responses using T , η , and μ .
 - (e) Hallucination Detection: After multiple iterations of monadic processing, check if categorical relationships still hold among responses. If any relationship deviates or breaks, flag the corresponding response as a potential hallucination.
 - (f) Human Validation: Submit flagged responses for human review to determine if they are genuine hallucinations.

[0079] 5. Outcome

- (a) The system will provide a refined set of ChatGPT responses, with potential hallucinations either flagged or filtered out, ensuring a more reliable and trustworthy user experience.

[0080] Multi-Agent Solvers

[0081] This subsection explains how to formulate basic mathematical problems. It also describes three common templates for formulating applied systems: linear programming, dynamic programming, and optimal control. Example formulations in the Prolog programming language are provided alongside the theoretical descriptions. Once the Prolog validation engine determines that ChatGPT's response has proposed a viable solution to a problem that can be solved by GPTProX's Multiple Agent Solvers, GPTProX will activate its own computational methods to calculate and provide the actual solution. It's important to note that ChatGPT's proposed solutions to math problems are based on language likelihoods, not direct calculation, so only the solutions computed by GPTProX can be relied upon as correct.

[0082] We use Matlab as demonstration of solvers. Other packages can be used similarly.

[0083] Basic Math Calculations

[0084] This section provides an overview of a Prolog setting designed to identify mathematical operations and subsequently call MATLAB to solve them.

[0085] Problem Formulation in Prolog

[0086] We first establish a Prolog setup for recognizing basic mathematical operations:

- Addition
- Multiplication
- Derivation
- Integration

[0087] The recognition is based on the use of definite clause grammar (DCG) in Prolog to parse and identify the operations.

[0088] Sample Code: Prolog Calls to MATLAB

[0089] The illustration of using Matlab solvers are based on experiences that they meet the necessary criteria. However, alternative solvers, including open-source ones, can be employed to potentially lower system costs.

: – use_module (library (process)).

solver_addition(X, Y, Result) : –
process_create (path (matlab), ...

solver_multiplication (X, Y, Result) : –
process_create (path (matlab) , ...

solver_derivation (Function , Variable , Result) : –
process_create (path (matlab) , ...

solver_integration (Function, Variable, Result) : –
process_create (path (matlab) , ...

[0090] Linear Programming (LP)

[0091] Linear programming is employed when addressing problems characterized by a linear objective function and linear constraints. Common applications include resource allocation, scheduling, and numerous other optimization tasks.

% Objective function

objective (Vars, Cost) : –
Cost is $C1 * X1 + C2 * X2 + \dots + CN * XN$.

% Constraints

constraint_1 (Vars) : –
 $A11 * X1 + A12 * X2 + \dots + A1N * XN \leq B1$.

constraint_2 (Vars): –
 $A21 * X1 + A22 * X2 + \dots + A2N * XN \leq B2$.

% Bounds

bounds(X1, Low1, Up1).
bounds(X2, Low2, Up2).

...
bounds(XN, LowN, UpN).

% Variable domain types

int (X1).
int (X2).

...
int (XN).

[0092] MATLAB Solver Call

[0093] Given the Prolog representation of the LP problem, we need to translate it into a MATLAB format and then call the 'linprog' function to solve it.

[0094] Dynamic Programming (DP)

[0095] Dynamic programming operates by fragmenting problems into tinier subproblems. Each subproblem is solved once, with its result stored, thus avoiding redundant computations. DP is instrumental for optimization problems where solutions can be recursively decomposed.

```
% Base case
solve( BaseCase, Solution).
```

```
% Recursive case
solve ( Problem , Solution): –
```

```
    decompose(Problem, SubProblem),
    solve(SubProblem, SubSolution),
    combine(SubSolution , Solution).
```

[0096] MATLAB Solver Call for Dynamic Programming

[0097] Given Prolog's recursive approach, translating this into MATLAB requires a bottom-up iterative method, where base cases are solved first and their solutions used to build up to the main problem.

[0098] Optimal Control

[0099] Optimal control endeavors to unearth control strategies that optimize specific performance indicators. It's particularly beneficial for systems spanning domains like robotics, economics, and numerous physical systems.

```
% Objective (typically a functional to be minimized/maximized over time)
objective(ControlSequence, Cost).
```

```
% System dynamics
dynamics (State, Control, NextState).
```

*% Constraints on control or state
constraints (State, Control).*

[00100] MATLAB Solver Call for Optimal Control

[00101] Given Prolog's formulation, translating this into MATLAB requires setting up a nonlinear optimization problem. The state dynamics, constraints, and objective function must be captured appropriately for MATLAB's solvers to be applied.

[00102] Multiple Agent

[00103] The GPTProX system leverages a modular, agent-based architecture for solving complex optimization problems requiring the integration of multiple techniques.

[00104] Each mathematical programming template, like linear programming or optimal control, is modeled as an independent agent. These agents are capable of solving standalone instances of their respective problem classes.

[00105] For situations requiring the combination of multiple optimization formulations, GPTProX coordinates the agents through constraint sharing. The state and decision variables from one agent are encoded as constraints for the others.

[00106] This allows the modular agents to collaborate towards an integrated solution. The agents exchange constraints over multiple interactions, iteratively refining each local solution to converge on a unified global optimization.

[00107] By decomposing multifaceted problems across specialized agents, and orchestrating their activities via constraints, GPTProX can tackle intricate real-world optimization challenges. This agent-based approach provides flexibility and scalability, extending the system's mathematical problem solving capacities through modular design.

[00108] Example: GPTProX in Post-Hurricane Aid

[00109] After hurricanes, utility companies face numerous unexpected challenges. Their usual rescue plans often fall short because each disaster brings unique problems. Residents in the affected areas might not get the specific help they need quickly using traditional methods.

[00110] GPTProX is a solution to this. It combines ChatGPT's knowledge with utility companies' specific data. When a resident asks for help, GPTProX consults

databases like the utility asset database, GIS, and ontology databases to give accurate and location-specific advice.

[00111] ChatGPT suggests possible responses based on the resident's query. These suggestions are then checked against the Prolog Knowledge Base to ensure the advice is relevant. If there are any doubts about the advice, the system uses Category theory to clear up any inconsistencies.

[00112] The system doesn't just give general information. It offers clear, actionable steps. For example, it can guide residents to safer areas, tell them how to access emergency supplies, or help them report specific damages for faster repairs.

[00113] The best feature of GPTProX is its Multi-Agent Optimization Solver. After a hurricane, it gives the most effective advice by considering all the various challenges. This means residents get smart, practical advice quickly.

[00114] Example: Personalized Health Assistance

[00115] GPTProX can be highly valuable in providing personalized health and wellness support by connecting its conversational capabilities with an individual's health data. For example, consider a user named John who regularly interacts with the system for diet and lifestyle suggestions.

[00116] John has built up a personal health knowledge base in GPTProX containing facts like:

- John has a peanut allergy
- John has hypertension and takes medication for it
- John's favorite foods are sushi, pasta, and salads
- John has a goal to exercise 3 times per week

[00117] When John asks GPTProX for meal recommendations, its conversation with ChatGPT generates responses with diet tips and recipes. The logic programming engine parses these responses into predicate logic for validation against John's personal knowledge base. Any contradictory information, like a recipe containing peanuts, is filtered out.

[00118] GPTProX also expands John's health knowledge by interfacing with his fitness tracker data and latest lab tests from his physician. It transforms this data into logic programming facts to keep the knowledge updated. Over time, GPTProX accumulates extensive knowledge about John's health profile.

[00119] The category theory component further vets the responses by modeling them as categorical objects mapped from John's input prompts. Deviations in the mappings allow detecting inconsistencies. For example, a dinner recipe mapped from a "nut-free" prompt containing nuts gets flagged.

[00120] For suggested actions like adopting an exercise regimen, the multi-agent optimization solver customizes the plan based on John's fitness levels, past injuries, exercise preferences, and weekly schedule.

[00121] In this manner, GPTProX delivers John personalized health guidance catered to his unique medical needs and constraints. The integration of the LLM with the health knowledge base makes the system more robust, accurate, and reliable.

[00122] Example: Software Project Bidding

[00123] GPTProX can assist a software company in formulating an optimal bid proposal for a large project. The company's knowledge base contains:

- Developer resources classified by skillsets (frontend, backend, DevOps etc.)
- Timeframe of resource availability
- Details of the company's flagship product and prior projects

[00124] When the high-level functionality requirements for the bid are entered into GPTProX, its conversation with ChatGPT suggests an implementation approach and required developer resources. The logic programming validator parses these suggestions to query the knowledge base. It verifies the feasibility per the company's current staffing and constraints. Unvalidated suggestions are discarded.

[00125] The category theory module further vets the implementation plan by modeling it against the requirements to catch inconsistencies. This enhances the robustness of the final proposal.

[00126] For the actions of assembling a technical team and estimating costs, GPTProX's optimization solver interfaces with the knowledge base. It generates an optimal allocation and schedule of developers based on skillsets, availability windows, and interdependencies.

[00127] These optimized outputs are combined to produce a comprehensive project execution plan. GPTProX runs financial calculations to attach detailed cost estimates to this plan.

[00128] The final output is an accurate bid proposal aligning staffing, timelines and budgets to the project's technical and functional needs. The knowledge base integration makes this feasible bid generation automated, rapid, and reliable.

[00129] Example: Personalized Vacation Planning

[00130] GPTProX can suggest customized vacation itineraries by incorporating a user's personal interests and preferences.

[00131] Consider Jack who maintains a knowledge base with facts like:

- Jack enjoys hiking, art galleries, and wine tasting
- Past vacation spots Jack liked were Barcelona and Florence
- Jack's budget for a 7-day trip is approximately \$3,000

[00132] When Jack asks GPTProX to plan a vacation in Paris, it constructs prompts infused with Jack's hobbies and prior vacation patterns. Chat-GPT generates Paris sightseeing recommendations based on these personalized prompts.

[00133] The logic programming module parses Chat-GPT's responses into Prolog format to query Jack's knowledge base. This filters out infeasible suggestions like expensive Michelin star restaurants.

[00134] GPTProX also interfaces with a Paris GIS system and travel ontology to pull in relevant geographical facts and semantic relationships. These expand the contextual grounding of the vacation plan.

[00135] The category theory validator models the vacation itinerary across multiple iterations, revising any inconsistencies violation Jack's constraints. [0154] Finally, the multi-agent optimization solver customizes the schedule based on opening hours, travel times between locations, crowd levels, and Jack's budget.

[00136] Example: Supply Chain Quick Adjustment

[00137] Introduction:

[00138] In today's dynamic business environment, unexpected disruptions in supply chains—be it due to natural disasters, geopolitical issues, or pandemics—can drastically affect a company's operations. Swift responses to such disruptions are essential to maintain operational efficiency. Leveraging ChatGPT, enhanced with GPTProX functionalities, can assist businesses in identifying and implementing alternative strategies at a rapid pace.

Actors:

1. Supply Chain Manager
2. ChatGPT with GPTProX
3. Local knowledge base

Preconditions:

- The company has an established supply chain management system.
- GPTProX with ChatGPT is integrated with the local knowledge base containing data about suppliers, logistics, inventory levels, and customer orders.

Scenario:

1. *Supply Chain Disruption:* An unexpected event (e.g., a major storm) disrupts the supply chain, halting shipments from a primary supplier.
2. *Consultation with GPTProX:* The Supply Chain Manager consults GPTProX regarding the issue, providing all necessary details.
3. *Local Knowledge Base Integration:*

- GPTProX sends query to ChatGPT for resolutions.
 - ChatGPT suggests a list of alternative suppliers and logistic partners that can potentially fill the gap.
 - GPTProX accesses the local knowledge base to understand the current state of inventory, pending orders, alternative suppliers, transport routes, and associated costs.
 - The knowledge base filters out unsuitable options based on current data—like blacklisted suppliers or routes with recent disruptions.
4. *Generation of Solutions:*
 - For each suggestion, a brief analysis of the pros, cons, and expected delivery timeframes is provided.
 - GPTProX uses its built-in optimizer to rank the solutions based on parameters such as cost-effectiveness, delivery speed, and reliability, ensuring the recommendations are not just feasible but also optimal.
 - GPTProX presents the optimized list of alternatives to the Supply Chain Manager, complete with actionable steps.
 5. *Implementation:* The company takes immediate action based on GPTProX's recommendations.
 6. *Feedback Loop:* Post-implementation results are fed back into the local knowledge base.

Postconditions:

- The supply chain disruption is addressed promptly.
- Customer satisfaction is maintained.

Benefits:

1. *Rapid Response:* Instant analysis and response facilitated by ChatGPT with GPTProX.
2. *Informed Decisions:* Data-driven recommendations ensure relevant and current decisions.
3. *Continuous Learning:* The system continuously learns and adapts through feedback.

[00139] Example: Fraud Detection in Phishing Emails

[00140] Advances in AI generation allow hackers to create customized phishing emails that appear authentic. Traditional spam filters relying on keywords are insufficient as generative AI can produce contextually coherent text. GTPProX's hybrid reasoning helps detect such sophisticated frauds.

[00141] Consider Sarah who maintains a personal knowledge base of facts like:

- Sarah banks with Chase and Wells Fargo
- Her payroll goes to Chase checking account
- She has an auto loan with Wells Fargo

[00142] When Sarah receives an email claiming her Wells Fargo account is locked, GTPProX parses it into logic predicates to check against her knowledge base. Statements contradicting Sarah's facts like "You have a wire transfer pending in your Bank of America account" are identified as hallucinations revealing the message as fraudulent.

[00143] For actionable claims like "Click here to unlock your account," GTPProX parses them into Prolog to send as prompts to ChatGPT. The risk assessment from ChatGPT determines actions like clicking links or downloading attachments as harmful.

[00144] By combining language understanding with logical reasoning, GTPProX provides robust protection against creatively crafted phishing attempts. The knowledge base integration grounds detections in the user's personal financial profile rather than relying solely on language patterns. This hybrid approach enhances fraud defense as AI generation grows more advanced.

[00145] Discussion

[00146] This example demonstrates how GTPProX can leverage logical and semantic inconsistencies to identify fraudulent emails generated by AI systems. The key aspects are:

- Personal knowledge base with facts about real accounts, assets, etc.
- Parsing emails into Prolog predicates to check against knowledge base
- Detecting factual contradictions reveals message as invalid
- Analyzing actions via ChatGPT identifies risks
- Custom protection based on individual's profile, not just language

[00147] Expanding this approach across an organization provides a robust defense by grounding detections in staff's collective factual knowledge. As creative phishing proliferates, reasoning-based techniques will become increasingly critical.

[00148] EXAMPLE IMPLEMENTATIONS

System for Validating and Enhancing Responses

1. A system for validating and enhancing responses from a large language model, comprising:
 1. A logic programming module configured to:
 - (a) Parse responses from the large language model into logic programming predicates;

A logic programming module configured to:
 2. (a) Query a logic programming knowledge base to validate the predicates;
 3. A logic programming module configured to:
 - (a) Filter out contradicted information from the responses.

Method for Enhancing Responses

- 2a. A computer-implemented method for enhancing responses from a large language model, comprising:
 1. Receiving a user input query;
 2. Obtaining one or more responses to the query from the large language model;

3. Parsing the responses into logic programming predicates;
4. Validating the predicates against a logic programming knowledge base;
5. Modeling the query and validated responses as categorical objects and morphisms;
6. Identifying semantic inconsistencies using category theory;
7. Optimizing any suggested actions in the responses using mathematical programming techniques; and
8. Outputting the validated and optimized responses.

Computer System for Enhancing Responses

- 2b. A computer system for enhancing responses from a large language model, comprising:
 1. A processor;
 2. A memory storing instructions executable by the processor to:
 - (a) Receive a user input query;
 - (b) Obtain responses from a large language model;
 - (c) Parse responses into logic programming predicates;
 - (d) Validate predicates against a knowledge base;
 - (e) Model responses categorically to identify inconsistencies;
 - (f) Optimize suggested actions mathematically;
 - (g) Output enhanced responses.

[00149] GLOSSARY

ChatGPT

ChatGPT A variation of the OpenAI GPT (Generative Pretrained Transformer) models designed specifically for conversational AI tasks.

GPT (Generative Pretrained Transformer) A series of AI models developed by OpenAI for natural language processing and understanding.

Transformer Architecture A deep learning model architecture introduced in the paper “Attention Is All You Need” by Vaswani et al., which forms the foundation for models like GPT and BERT.

Fine-tuning The process of training a pre-trained model on a specific dataset to adapt it for particular tasks, such as conversational interactions.

LLM (Language Model)

LLM Short for “Language Model”, it’s a type of artificial intelligence model that understands and generates human-like text.

Token A unit of text, which can be as short as one character or as long as one word, that language models read.

Zero-shot, One-shot, and Few-shot Learning Terms that describe how a model performs tasks with no examples, one example, or a few examples, respectively.

Prolog

Prolog (Programming in Logic) A high-level programming language primarily associated with artificial intelligence and symbolic reasoning.

Fact A basic statement in Prolog representing knowledge.

Rule A statement in Prolog that expresses a relationship between facts.

Backtracking Prolog’s method of searching through possible solutions to find one that satisfies a given query.

Predicate A function that returns a boolean value (true or false), used in Prolog to define relations among objects.

Horn Clause Horn clauses are a type of logical expression used in logic programming and formal logic, consisting of a disjunction of literals with at most one positive literal. They are commonly used in Prolog to represent rules and facts, enabling efficient reasoning and inference.

MATLAB

MATLAB (Matrix Laboratory) A high-level programming language and environment designed for numerical and matrix computations, data analysis, and visualization.

Simulink An add-on product for MATLAB, it provides a graphical environment for modeling, simulating, and analyzing dynamic systems.

M-file A script or function file written in MATLAB’s native language.

Matrix A rectangular array of numbers, often used in MATLAB for computations and data representation.

SpaCy

SpaCy An open-source library for natural language processing in Python.

Tokenization The process of splitting a text into words, phrases, symbols, or other meaningful elements called tokens.

NER (Named Entity Recognition) The process of identifying and classifying named entities in text, such as person names, organizations, and locations.

Dependency Parsing Analyzing the grammatical structure of a sentence to determine the relationships between words.

Pipeline A series of processing steps, often used in SpaCy to process text through various stages like tokenization, tagging, and parsing.

[00150] Exemplary Computing Device: Referring to FIG. 7, a computing device 1200 is illustrated which may can be configured, via one or more of an application 1211 or computer-executable instructions, to execute functionality described herein. More particularly, in some embodiments, aspects of the methods herein may be translated to software or machine-level code, which may be installed to and/or executed by the computing device 1200 such that the computing device 1200 is configured to execute functionality described herein. It is contemplated that the computing device 1200 may include any number of devices, such as personal computers, server computers, hand-held or laptop devices, tablet devices, multiprocessor systems, microprocessor-based systems, set top boxes, programmable consumer electronic devices, network PCs, minicomputers, mainframe computers, digital signal processors, state machines, logic circuitries, distributed computing environments, and the like.

[00151] The computing device 1200 may include various hardware components, such as a processor 1202, a main memory 1204 (e.g., a system memory), and a system bus 1201 that couples various components of the computing device 1200 to the processor 1202. The system bus 1201 may be any of several types of bus structures including a memory bus or memory controller, a peripheral bus, and a local bus using any of a variety of bus architectures. For example, such architectures may

include Industry Standard Architecture (ISA) bus, Micro Channel Architecture (MCA) bus, Enhanced ISA (EISA) bus, Video Electronics Standards Association (VESA) local bus, and Peripheral Component Interconnect (PCI) bus also known as Mezzanine bus.

[00152] The computing device 1200 may further include a variety of memory devices and computer-readable media 1207 that includes removable/non-removable media and volatile/nonvolatile media and/or tangible media, but excludes transitory propagated signals. Computer-readable media 1207 may also include computer storage media and communication media. Computer storage media includes removable/non-removable media and volatile/nonvolatile media implemented in any method or technology for storage of information, such as computer-readable instructions, data structures, program modules or other data, such as RAM, ROM, EEPROM, flash memory or other memory technology, CD-ROM, digital versatile disks (DVD) or other optical disk storage, magnetic cassettes, magnetic tape, magnetic disk storage or other magnetic storage devices, or any other medium that may be used to store the desired information/data and which may be accessed by the computing device 1200. Communication media includes computer-readable instructions, data structures, program modules, or other data in a modulated data signal such as a carrier wave or other transport mechanism and includes any information delivery media. The term "modulated data signal" means a signal that has one or more of its characteristics set or changed in such a manner as to encode information in the signal. For example, communication media may include wired media such as a wired network or direct-wired connection and wireless media such as acoustic, RF, infrared, and/or other wireless media, or some combination thereof. Computer-readable media may be embodied as a computer program product, such as software stored on computer storage media.

[00153] The main memory 1204 includes computer storage media in the form of volatile/nonvolatile memory such as read only memory (ROM) and random access memory (RAM). A basic input/output system (BIOS), containing the basic routines that help to transfer information between elements within the computing device 1200 (e.g., during start-up) is typically stored in ROM. RAM typically contains data and/or program modules that are immediately accessible to and/or presently being operated on by processor 1202. Further, data storage 1206 in the form of Read-Only

Memory (ROM) or otherwise may store an operating system, application programs, and other program modules and program data.

[00154] The data storage 1206 may also include other removable/non-removable, volatile/nonvolatile computer storage media. For example, the data storage 1206 may be: a hard disk drive that reads from or writes to non-removable, nonvolatile magnetic media; a magnetic disk drive that reads from or writes to a removable, nonvolatile magnetic disk; a solid state drive; and/or an optical disk drive that reads from or writes to a removable, nonvolatile optical disk such as a CD-ROM or other optical media. Other removable/non-removable, volatile/nonvolatile computer storage media may include magnetic tape cassettes, flash memory cards, digital versatile disks, digital video tape, solid state RAM, solid state ROM, and the like. The drives and their associated computer storage media provide storage of computer-readable instructions, data structures, program modules, and other data for the computing device 1200.

[00155] A user may enter commands and information through a user interface 1240 (displayed via a monitor 1260) by engaging input devices 1245 such as a tablet, electronic digitizer, a microphone, keyboard, and/or pointing device, commonly referred to as mouse, trackball or touch pad. Other input devices 1245 may include a joystick, game pad, satellite dish, scanner, or the like. Additionally, voice inputs, gesture inputs (e.g., via hands or fingers), or other natural user input methods may also be used with the appropriate input devices, such as a microphone, camera, tablet, touch pad, glove, or other sensor. These and other input devices 1245 are in operative connection to the processor 1202 and may be coupled to the system bus 1201, but may be connected by other interface and bus structures, such as a parallel port, game port or a universal serial bus (USB). The monitor 1260 or other type of display device may also be connected to the system bus 1201. The monitor 1260 may also be integrated with a touch-screen panel or the like.

[00156] The computing device 1200 may be implemented in a networked or cloud-computing environment using logical connections of a network interface 1203 to one or more remote devices, such as a remote computer. The remote computer may be a personal computer, a server, a router, a network PC, a peer device or other common network node, and typically includes many or all of the elements described

above relative to the computing device 1200. The logical connection may include one or more local area networks (LAN) and one or more wide area networks (WAN), but may also include other networks. Such networking environments are commonplace in offices, enterprise-wide computer networks, intranets and the Internet.

[00157] When used in a networked or cloud-computing environment, the computing device 1200 may be connected to a public and/or private network through the network interface 1203. In such embodiments, a modem or other means for establishing communications over the network is connected to the system bus 1201 via the network interface 1203 or other appropriate mechanism. A wireless networking component including an interface and antenna may be coupled through a suitable device such as an access point or peer computer to a network. In a networked environment, program modules depicted relative to the computing device 1200, or portions thereof, may be stored in the remote memory storage device.

[00158] Certain embodiments are described herein as including one or more modules. Such modules are hardware-implemented, and thus include at least one tangible unit capable of performing certain operations and may be configured or arranged in a certain manner. For example, a hardware-implemented module may comprise dedicated circuitry that is permanently configured (e.g., as a special-purpose processor, such as a field-programmable gate array (FPGA) or an application-specific integrated circuit (ASIC)) to perform certain operations. A hardware-implemented module may also comprise programmable circuitry (e.g., as encompassed within a general-purpose processor or other programmable processor) that is temporarily configured by software or firmware to perform certain operations. In some example embodiments, one or more computer systems (e.g., a standalone system, a client and/or server computer system, or a peer-to-peer computer system) or one or more processors may be configured by software (e.g., an application or application portion) as a hardware-implemented module that operates to perform certain operations as described herein.

[00159] Accordingly, the term “hardware-implemented module” encompasses a tangible entity, be that an entity that is physically constructed, permanently configured (e.g., hardwired), or temporarily configured (e.g., programmed)

to operate in a certain manner and/or to perform certain operations described herein. Considering embodiments in which hardware-implemented modules are temporarily configured (e.g., programmed), each of the hardware-implemented modules need not be configured or instantiated at any one instance in time. For example, where the hardware-implemented modules comprise a general-purpose processor configured using software, the general-purpose processor may be configured as respective different hardware-implemented modules at different times. Software may accordingly configure the processor 1202, for example, to constitute a particular hardware-implemented module at one instance of time and to constitute a different hardware-implemented module at a different instance of time.

[00160] Hardware-implemented modules may provide information to, and/or receive information from, other hardware-implemented modules. Accordingly, the described hardware-implemented modules may be regarded as being communicatively coupled. Where multiple of such hardware-implemented modules exist contemporaneously, communications may be achieved through signal transmission (e.g., over appropriate circuits and buses) that connect the hardware-implemented modules. In embodiments in which multiple hardware-implemented modules are configured or instantiated at different times, communications between such hardware-implemented modules may be achieved, for example, through the storage and retrieval of information in memory structures to which the multiple hardware-implemented modules have access. For example, one hardware-implemented module may perform an operation, and may store the output of that operation in a memory device to which it is communicatively coupled. A further hardware-implemented module may then, at a later time, access the memory device to retrieve and process the stored output. Hardware-implemented modules may also initiate communications with input or output devices.

[00161] Computing systems or devices referenced herein may include desktop computers, laptops, tablets e-readers, personal digital assistants, smartphones, gaming devices, servers, and the like. The computing devices may access computer-readable media that include computer-readable storage media and data transmission media. In some embodiments, the computer-readable storage media are tangible

storage devices that do not include a transitory propagating signal. Examples include memory such as primary memory, cache memory, and secondary memory (e.g., DVD) and other storage devices. The computer-readable storage media may have instructions recorded on them or may be encoded with computer-executable instructions or logic that implements aspects of the functionality described herein. The data transmission media may be used for transmitting data via transitory, propagating signals or carrier waves (e.g., electromagnetism) via a wired or wireless connection.

[00162] The described methods, processes, operations, and associated actions may also be performed in various orders in addition to the order described in this application, in parallel, and/or simultaneously. The described systems are exemplary in nature and may include additional elements and/or omit elements. Furthermore, references to or “one example” of the present disclosure are not intended to be interpreted as excluding the existence of additional embodiments that also incorporate the recited features. It will be understood that when a certain part or process “includes” a certain component or operation, that part or process does not exclude another component or operation. While illustrative examples of the name screening techniques using phonetic embeddings have been described herein including systems, devices, and the like, it is to be understood that various other adaptations and modifications may be made within the spirit and the scope of the examples herein.. Additionally, it is appreciated that while specific graphics are shown and described, such graphics are illustrative and exemplary and are not intended to limit the scope of this disclosure.

[00163] The foregoing description has been directed to specific examples. It will be apparent, however, that other variations and modifications may be made to the described examples, with the attainment of some or all of their advantages. For instance, it is expressly contemplated that the components and/or elements described herein can be implemented as software being stored on a tangible (non-transitory) computer-readable medium, devices, and memories (e.g., disks/CDs/RAM/EEPROM/etc.) having program instructions executing on a computer, hardware, firmware, or a combination thereof. Further, methods describing the various functions and techniques described herein can be implemented using computer-

executable instructions that are stored or otherwise available from computer readable media. Such instructions can comprise, for example, instructions and data which cause or otherwise configure a general-purpose computer, special purpose computer, or special purpose processing device to perform a certain function or group of functions. Portions of computer resources used can be accessible over a network. The computer executable instructions may be, for example, binaries, intermediate format instructions such as assembly language, firmware, or source code. Examples of computer-readable media that may be used to store instructions, information used, and/or information created during methods according to described examples include magnetic or optical disks, flash memory, USB devices provided with non-volatile memory, networked storage devices, and so on. In addition, devices implementing methods according to these disclosures can comprise hardware, firmware and/or software, and can take any of a variety of form factors. Typical examples of such form factors include laptops, smart phones, small form factor personal computers, personal digital assistants, and so on. Functionality described herein also can be embodied in peripherals or add-in cards. Such functionality can also be implemented on a circuit board among different chips or different processes executing in a single device, by way of further example. Instructions, media for conveying such instructions, computing resources for executing them, and other structures for supporting such computing resources are means for providing the functions described in these disclosures. Accordingly, this description is to be taken only by way of example and not to otherwise limit the scope of the examples herein. Therefore, it is the object of the appended claims to cover all such variations and modifications as come within the true spirit and scope of the examples herein.

[00164] Additional Examples

[00165] The present inventive disclosure details GPTProX, an artificial intelligence system integrating large language models with logic programming and mathematical optimization to deliver precise, customized enterprise solutions. GPTProX overcomes unreliable hallucinated responses from systems like ChatGPT through a multi-pronged approach. A logic programming engine parses responses into verifiable logic predicates to detect contradictions against a real-world knowledge base.

Additionally, a category theory module models responses as mathematical morphisms, identifying semantic inconsistencies that may indicate false information.

[00166] For actionable queries, GPTProX leverages problem templates for techniques including linear programming, dynamic programming, and optimal control. It decomposes objectives into modular components solvable via multi-agent optimization. Combining these solutions generates optimized, executable directives tailored to the user.

[00167] Knowledge representation is enriched by interfacing with diverse structured data sources such as ontologies, geographic information systems, and financial databases. The logic programming knowledge base also expands through inductive learning.

[00168] This fusion of language modeling, logical reasoning, category theory, and mathematical optimization yields an enterprise-grade AI assistant. GPTProX overcomes the accuracy pitfalls of large language models to deliver customizable, logically valid, and real-world optimized responses across industries. The system's principled architecture brings precision and robust configurability to conversational AI.

[00169] In some examples, GPTProX uses multi-agent solving to supplement the LLM response. In a **manufacturing** scenario, one agent could use **linear programming** to optimize the allocation of raw materials to different production lines, ensuring the most efficient use of resources. Another agent could employ **dynamic programming** to manage sequential decision-making around production schedules, while a third agent might use **optimal control** to determine the best strategies over time for minimizing energy consumption during the production process. Each agent collaborates by sharing relevant constraints (e.g., availability of machines, delivery deadlines), and their combined solutions are processed to achieve an optimal global result.

[00170] Other non-limiting examples are contemplated, including the following.

- **API integration:** This addition specifies that the constructed input messages are formatted into API calls, which is a common method for interacting with enterprise systems.

- **Practical applicability:** By explicitly mentioning API calls, this claim enhances clarity by reflecting the standard communication between generative AI systems (such as GPTProX) and existing enterprise systems.

[00171] It should be understood from the foregoing that, while particular embodiments have been illustrated and described, various modifications can be made thereto without departing from the spirit and scope of the invention as will be apparent to those skilled in the art. Such changes and modifications are within the scope and teachings of this invention as defined in the claims appended hereto.

CLAIMS

What is claimed is:

1. A method for enhancing responses from a large language model (LLM), comprising:
 - (a) accessing at least one response to a query from a large language model;
 - (b) parsing the at least one response into logic programming predicates defining a plurality of facts and actions associated with the at least one response;
 - (c) conducting an accuracy (correctness) validation operation for the at least one response in view of a knowledge base for logic programming including:
 - (c)(i) verifying and maintaining facts of the plurality of facts in the at least one response shown to be true via the knowledge base, and
 - (c)(ii) removing any other facts of the plurality of facts from the at least one response defining negations shown to be true in the knowledge base; and
 - (d) generating an output evaluating and returning a validated version of the at least one response based on the accuracy validation operation.
2. The method of claim 1, further comprising:
 - determining the plurality of facts cannot be verified by the knowledge base; and
 - conducting category validation, including:
 - generating one or more prompts that altered from the query but are semantically equivalent to the query,

for each of the one or more prompts, obtaining a respective response from the large language model,
establishing a structural relationship between the generated prompts and the corresponding responses from the large language model, and
detecting for any semantic inconsistencies using category theory, wherein an identification of a semantic inconsistency reflects indicates a lack of reliability associated with the at least one response.

3. The method of claim 1, wherein the logic programming predicates are defined from logical programming sentences associated with the at least one response and include conjunctions to structure relationships between the logical programming sentences.
4. The method of claim 1, further comprising:
conducting multi-agent optimization solving for supplementing the at least one response with additional functionality or information associated with the query.
5. The method of claim 4, further comprising:
iterating the large language model to assist in formulating at least one mathematical model to customize a modified version of the response based at least in part on the query.
6. The method of claim 4, further comprising supplementing the at least one response, by:
generating a plurality of agents, each of the plurality of agents configured for solving a respective optimization problem associated with the query using linear programming, dynamic programming, or optimal control, and
assigning objective functions and constraints for each agent.

7. The method of claim 6, wherein each agent of the plurality of agents exchanges constraints and decisions with other agents over multiple iterations, under the guidance of the LLM, until convergence on a globally optimized solution.
8. The method of claim 1, further comprising:
- parsing the at least one response into the logic programming predicates to construct input messages for an enterprise computer system;
 - integrating input messages as constructed with the existing enterprise computer systems, including informational systems and control systems, by formatting the input messages into application programming interface (API) calls for communication with said existing enterprise computer systems; and
 - distributing the validated version of the at least one response to said existing enterprise computer systems via APIs, enabling automated data queries and control actions based on generative artificial intelligence outputs.