Automated Selection of Processing Functions in Response to User Message

FIELD

Described embodiments lie in the field of responding to messages instructing a software service to execute processing on behalf of a user. In particular, embodiments relate to leveraging appropriately constructed LLM prompts to determine how to respond to a message received from a user.

INTRODUCTION

Software service end users may be required to perform certain functions with a frequency that can cause the functions to become onerous. Furthermore, each time the function is to be performed the end user may be required to login to the software service, and navigate s system of interfaces and menus to reach the function, which causes further inconvenience.

Software services, such as subscription software services, commonly provide end users with automated personal assistant type services, that can provide information and processing functions to users in response to natural language commands.

The automated personal assistants need to be trained, requiring the software service provider to generate large training datasets and to implement a training process.

Functionality of such automated personal assistants is enhanced by granting freedom to the automated personal assistants in terms of responding to user messages. For example, such freedom may include leveraging generative AI to determine responses, rather than relying on predefined processing logic.

Owing to the nature of generative AI, and its tendency to hallucinate, it is desirable to guardrail the generative AI in responding to user messages.

It is desirable to improve user experience by at least partially ameliorating some of the inconvenience associated with the manual execution of common tasks. It is desirable to at least partially ameliorate the overheads associated with the training of an automated assistant at the software service.

SUMMARY

Some embodiments relate to a computer-implemented method for execution by a software service provider providing a software service to an end user, the method comprising: receiving, via a user interface, message content from an end user expressing an intent; generating a search string based on the received message content; inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string; generating and outputting to a first LLM, an LLM prompt comprising an expression of user intent from the received message content, and the text descriptors of only a subset of the set of processing functions, the subset being the members of the set of processing functions of which the text descriptors were ranked in the top n results for semantic proximity to the search string by the searching mechanism, n being a positive integer equal to or greater than 2, and requesting that the first LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message content; causing the selected processing function to execute.

The user interface may include a messaging interface for exchanging messages with a virtual assistant. The message content from the end user expressing the intent may be a message or messages in a message conversation between the end user and the virtual assistant. Generating the search string based on the received message content may comprise: generating a further LLM prompt comprising the received message content and one or more additional excerpts from the message conversation, and requesting an expression of user intent be extracted from the message content and one or more additional excerpts; outputting the generated further LLM prompt to a second LLM; parsing the reply from the second LLM into the search string.

The further LLM prompt may also request that the second LLM augment the expression of user intent with contextual information for interpreting the expression of user intent, wherein the contextual information includes one or more from among:

- an indication of a geographical region relevant to the end user;
- an indication of a domain of knowledge relevant to the end user;
- an indication of a field of commercial activity relevant to the end user;
- identifying information relevant to the end user.

The first LLM and the second LLM may be the same LLM; or the first LLM and the second LLM may be hosted by distinct service providers.

In some embodiments, the searching mechanism comprises: a vector space in which vectors representing the text descriptors of each member of the set of processing functions are expressed; a vectorizing model configured to transform the input search string from a string format to a vector representing the input search string expressed in the vector space; wherein the searching mechanism is configured to use vector distance between the vector representing the input search string and the vectors representing the text descriptors of each member of a set of processing functions, respectively, to rank the text descriptors of each member of a set of processing functions according to semantic proximity to the search string.

In some embodiments, the searching mechanism is a full text search of the search string against the text descriptors of each member of the set of processing functions; wherein the searching mechanism is configured to rank the text descriptors of each member of a set of processing functions according to semantic proximity to the search string based on results of the full text search.

The searching mechanism may be hosted by the software service provider as a function of, or otherwise accessible to, the user interface; or the searching mechanism may be hosted by a third party and is accessible to the software service provider as an on-demand service.

In some embodiments, with respect to either one, or both, of the first LLM and the second LLM: the LLM is hosted by the software service provider and has been at least partially trained with training data including interface messaging histories of end users of the software service using the user interface or the messaging interface; or the LLM is a generic LLM hosted by a third party and responding to received LLM prompts as an on-demand service to the software service provider.

In some embodiments, the computer-implemented method further comprises: outputting, via the user interface or messaging interface, a reply to the received message content, the reply comprising, or representing, a result of the execution of the selected processing function.

The message content may comprise a sequence of messages from the end user; wherein the search string includes elements from a plurality of different messages in the sequence.

In some embodiments, the computer-implemented method further comprises: following the return of the selected processing function from the first LLM and prior to or during the execution of the selected processing function, outputting, via the user interface or messaging interface, a confirmatory query to the end user requesting confirmation that the end user intends the selected processing function to be executed; receiving a positive response to the confirmatory query from the end user; augmenting the text descriptor of the selected processing function with the received message content from the end user expressing the intent.

In some embodiments, the LLM prompt further comprises, or a new LLM prompt for a third LLM, being the same as the first and second LLMs or different from the first and second LLMs, is created during the execution of the processing function and comprises: a request that the LLM extract, from the received message content, values for each of one or more data fields identified in the text descriptor of the selected processing function, and return the extracted values in response to the LLM prompt; a member of the set of processing functions is a processing function comprising a plurality of different sequences of component actions, wherein causing the selected processing function to execute includes selecting one from among the plurality of different sequences, and causing the processing function to execute in accordance with the selected sequence; and the selection of one from among the plurality of different sequences is based on one or more from among: for which of the one or more data fields identified in the text descriptor of the selected processing function the LLM was able to extract a value; for which of the one or more data fields identified in the text descriptor of the selected processing function the LLM was not able to extract a value.

The computer-implemented method may further comprise: following the selection of the sequence and prior to or during the execution of the selected processing function in accordance with the selected sequence of component actions, outputting, via the user interface or messaging interface, a confirmatory query to the end user requesting confirmation that the end user intends the selected processing function to be executed in accordance with the selected sequence of component actions; in response to receiving a positive response to the confirmatory query from the end user, causing the processing function to execute in accordance with the selected sequence of component actions; and in response to receiving a negative response to the confirmatory query from the end user, making a new selection of a sequence from among the plurality of different sequences, and causing the processing function to execute in accordance with the new selected sequence of component actions.

In some embodiments, the selected sequence of component actions includes an API call, and causing the processing function in accordance with the selected sequence includes making the API call using one or more of the data field values extracted by the LLM as a parameter.

In some embodiments, the causing the selected processing function to execute comprises determining and following an execution traversal path in a data graph; wherein the data graph is a directed data graph representing the processing function, each of the component actions being represented by a graph node, and each of the plurality of different executable sequences of component actions being represented by a traversal path connecting graph nodes representing component actions in the executable sequence between a traversal path end node and a traversal path start node; and wherein causing the selected processing function to execute comprises performing the component actions represented by graph nodes on the execution traversal path between the traversal path end node and the traversal path start node in forward or reverse sequence, starting at either the traversal path start node or the traversal path end node, and determining the execution traversal path during the execution by, when a plurality of valid next nodes in the execution traversal path are available, selecting which of the next nodes to include in the execution traversal path in dependence upon an outcome of a component action already performed in the execution traversal path or in dependence upon information extracted from the received message content.

In some embodiments, each of the set of processing functions maps to a different graph traversal path end node, and the execution traversal path is at least partially determined by identifying the graph traversal path end node mapped to by the selected processing function, and finding a starting node from which there are valid graph traversal paths terminating at the identified graph traversal path end node, and determining the execution traversal path between the starting node and the identified graph traversal path end node along one of the valid graph traversal paths.

In some embodiments, causing the selected processing function to execute further comprises determining the execution traversal path during the execution by, when the traversal path is at a graph node from which two or more directed edges originate or at which two or more directed edges terminate, selecting which directed edge to follow in dependence upon a data field value extracted from the received message. Selecting which directed edge to follow in dependence upon a data field value extracted from the received message may be dependent upon the data

field instantiated by the data field value, wherein the data field is one of a plurality of data fields defined in a text descriptor of the processing function, each of the plurality of data fields corresponding to a different directed edge originating from, or terminating at, a single graph node in the data graph.

In some embodiments, the user interface is a messaging interface of a virtual assistant, and wherein a component action represented by a graph node on the traversal path is a data retrieval component action, and wherein plural directed edges originate or terminate at the graph node representing the data retrieval component action, one of the plural directed edges being selectable in determining the execution traversal path when a number of retrieved data artefacts exceeds a predefined threshold maximum number and terminating at a graph node representing a query user component action, the query user component action comprising outputting a query to a user via the virtual assistant to select a subset of retrieved data artefacts corresponding in size to the predefined threshold maximum number, and receiving an indication of a selected subset from the user in response to the query.

Determining the execution traversal path may comprise following one or more directed edges to return to the graph node representing the data retrieval component action, and selecting which of plural directed edges originating or terminating at the graph node representing the data retrieval component action to follow in dependence upon the selected subset indicated by the user in response to the output query.

The user interface may be a messaging interface of a virtual assistant, the virtual assistant being configured, following the selecting a processing function to execute in response to the received message, to output a confirmatory message to the end user via the messaging interface identifying or describing the selected processing function, and seeking a positive or negative reply, the virtual assistant being configured to respond to a positive reply by proceeding with causing the selected processing function to execute, and being configured to respond to a negative reply by obtaining a new expression of user intent and repeating the selecting with the new expression of user intent.

In some embodiments, performing the component actions represented by graph nodes on the execution path in sequence includes performing one or more component actions comprising a data extraction task, the data extraction task comprising extracting information from the

message content or from data stored by the software service and accessible to the end user or relating to the end user, the computer implemented method further comprising outputting to the end user via the user interface the extracted information and seeking a positive reply indicating approval of the extracted information, or a negative reply indicating disapproval of the extracted information, and in the event of receiving the positive reply, proceeding along the execution traversal path, and in the event of receiving the negative reply, either repeating the data extraction task with different parameters or runtime variables, or obtaining replacement information from a message input to the user interface by the end user.

In some embodiments, the user interface is a virtual assistant, or is a component of a virtual assistant, and wherein a component action represented by a graph node on the traversal path is a data retrieval component action, and wherein plural directed edges originate or terminate at the graph node representing the data retrieval component action, one of the plural directed edges being selectable in determining the execution traversal path when a number of retrieved data artefacts does not meet a predefined threshold minimum number and terminating at a graph node representing a query user component action, the query user component action comprising outputting a query to a user via the virtual assistant to initiate creation of one or more data artefacts to meet the predefined threshold minimum number, or to change a parameter in the data retrieval action, and a reply from the user in response to the query.

Determining the execution traversal path may comprise following one or more directed edges to return to the graph node representing the data retrieval component action, and selecting which of plural directed edges originating or terminating at the graph node representing the data retrieval component action to follow in dependence upon reply from the user in response to the output query.

In some embodiments, causing the selected processing function to execute further comprises determining the execution traversal path during the execution by, when the traversal path is at a graph node from which two or more directed edges originate or at which two or more directed edges terminate, selecting which of the two or more directed edges to follow in dependence upon a user intent extracted from the received message.

In some embodiments, the starting graph node is a predefined starting graph node for the processing function, or the starting graph node is selected in dependence upon a user intent extracted from the received message.

In some embodiments, the user intent is extracted from the received message by submitting the received message to an LLM in a prompt requesting that the LLM select, from a plurality of processing functions described in the prompt, a processing function best matching the user intent.

Some embodiments relate to an apparatus including a computer system comprising processor hardware and memory hardware, the computer system operable to provide a software service to an end user, the memory hardware storing processing instructions which, when executed by the processor hardware, cause the computer system to perform a method including: receiving, via a user interface, message content from an end user expressing an intent; generating a search string based on the received message content; inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string; generating and outputting to a first LLM, an LLM prompt comprising an expression of user intent from the received message content, and the text descriptors of only a subset of the set of processing functions, the subset being the members of the set of processing functions of which the text descriptors were ranked in the top n results for semantic proximity to the search string by the searching mechanism, n being a positive integer equal to or greater than 2, and requesting that the first LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message content; causing the selected processing function to execute.

Some embodiments relate to an apparatus including a computer system comprising processor hardware and memory hardware, the computer system operable to provide a software service to a plurality of end users, the memory hardware storing processing instructions which, when executed by the processor hardware, cause the computer system to perform a computer-implemented method according to any one of the described embodiments.

Some embodiments relate to a computer program comprising processing instructions which, when executed by processor hardware in a computing device, cause the computing device to perform a method including: receiving, via a user interface, message content from an end user

expressing an intent; generating a search string based on the received message content; inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string; generating and outputting to a first LLM, an LLM prompt comprising an expression of user intent from the received message content, and the text descriptors of only a subset of the set of processing functions, the subset being the members of the set of processing functions of which the text descriptors were ranked in the top n results for semantic proximity to the search string by the searching mechanism, n being a positive integer equal to or greater than 2, and requesting that the first LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message content; causing the selected processing function to execute.

Some embodiments relate to a computer program comprising processing instructions which, when executed by processor hardware in a computing device, cause the computing device to perform a computer-implemented method according to any of the described methods.

Some embodiments relate to a non-transient computer-readable medium storing any of the described processing instructions.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments are set out in the detailed description below, with reference to the accompanying drawings in which:

Figure 1A illustrates a computer-implemented method;

Figure 1B illustrates a computer-implemented method;

Figure 2 illustrates a data graph;

Figure 3 illustrates a data graph;

Figure 4 illustrates a computer-implemented method;

Figures 5A to 5D illustrate computer-implemented methods;

Figure 6 illustrates an apparatus;

Figure 7 is a schematic illustration of a system providing a software service; and

Figure 8 illustrates a computer-implemented method.

DETAILED DESCRIPTION

Figure 1A illustrates a computer-implemented method. At step S101 message content is received at the software service from an end user. The message content may be a single message or a sequence of messages. The sequence of messages may be, or may be part of, a messaging conversation between the end user and a virtual assistant. The virtual assistant being functionality underpinning a messaging interface.

The software service may be hosted by one or more servers, which servers may implement one or more virtual servers. The servers may themselves store, or may be in data communication with one or more data storage servers which store end user data specific to each end user of the software service. Access to such data is controlled, for example by end user identity, access level, organization, etc. For example, there is no data sharing across different organisations. However, end users belonging to the same organization may have access to common data, wherein access to those data may be managed based on the identity of the particular end user and parameters such as the team to which the end user belongs within the organization, level of seniority within a defined hierarchy, etc.

The software service may be an online accounting software service providing user functionality relating to accountancy and bookkeeping, including contact information storage and management, invoicing, reporting, and generation of other financial documents such as purchase orders. End user content is a term employed to refer to documentation and database entries stored and maintained by the software service and accessible to the pertinent end user. The end user content may have been generated by the software service according to inputs from the end user themselves or another end user belonging to the same organization (e.g. where the software service enables data sharing between multiple users belonging to the same organization). The software service provides data storage and processing functions to registered end users. User registration means that each different end user has a unique ID within the software service.

The software service may be entirely online so that its functionality is accessed via a web browser. Alternatively or additionally, an end user application for computers, tablets, mobile phones etc, may provide at least part of the functionality of the software service, and/or one or more user interfaces for accessing the functionality of the software service.

The message content may be an email, an SMS, a whatsapp message, a voice note, or some other medium for electronic transfer of information from an end user to the software service. The message may be to a generic mailbox at the software service, with the software service identifying the end user among a population of end users based on a lookup of the sender details in the message metadata in a database of end user sender details. Alternatively each end user may have a dedicated mailbox at the software service to which to address messages. In the case of a voice note, the receive step may include a speech to text process at the software service to transform the voice note from audio data to text data.

The message content may be to a chatbot, AI assistant, virtual assistant, or equivalent (which may be collectively referred to as a virtual assistant). The AI assistant, chatbot, or virtual assistant, may leverage generative AI to phrase responses to messages input by an end user. Content for inclusion in (either directly, indirectly via a link, or indirectly via description) the response to the end user may be phrased by the chatbot, AI assistant, virtual assistant, or equivalent, using content generated by the software service in performing a processing function such as at S105.

The messaging interface may be setup for the sole purpose of receiving instructions from end users (for example an email inbox dedicated to receiving processing instructions from one or a plurality of users) or a virtual assistant, so that any received message is assumed to contain an instruction (it being implicit that a user content processing function is intended) and so to cause execution of the method of Figure 1A.

In another alternative, an initial filtering step or equivalent may be performed to determine whether or not the received message content includes a processing instruction or equivalent, and to proceed with the remainder of the method if the message content does include the processing instruction or equivalent, and not otherwise. The filtering may be performed by intent extraction performed in a deterministic way according to a set of processing rules (in a simple example, does the message content include a question or a word typically in an instruction or command such as ‘please’, ‘would you’, or one of a predefined list of verbs associated with the available user content processing functions). Or conversely, messages can be filtered out where they can be identified as expressing thanks for a previously-performed processing function. Filtering is taken to mean determining whether or not to continue with processing the received message content according to steps S102 onward.

The message content is received by the software service from the end user via a user interface. The user interface may be or include a messaging interface. The user interface may be a speech-to-text interface that receives audio data and outputs a text representation of speech detected in the audio data. The output of the text representation of detected speech may be output to a messaging interface or directly to a virtual assistant. The messaging interface may be provided by the software service directly such as a virtual assistant, or may be external to the messaging interface such as a generic emailing interface or messaging interface such as SMS or whatsapp. The messaging interface may comprise an email inbox at the software service, and the message content may comprise an email being an individual email message or a thread of email messages. Alternatively or additionally, the messaging interface may comprise an SMS or whatsapp inbox (at the software service provider), and the message content is an SMS or whatsapp message or a series of whatsapp messages or SMS messages. Such messages may comprise text or may be audio messages which are transformed to text in a preprocessing step at the software service. The messaging interface may be a virtual assistant interface conducting a multi-message text or speech conversation with the end user.

Virtual assistant is understood to mean a messaging interface enabling an end user to input messages to the software service, and underlying processing logic and processing routines to cause the software service to respond to the input messages, whether that responding is by processing (executing a processing function) data, and/or outputting a message in reply to the received message.

The message content may comprise natural language content. The messaging interface may be a virtual assistant comprising a natural language messaging interface for exchanging text messages in a session between the end user and the virtual assistant (being a component of the software service), and may be a single-turn or a multi-turn messaging interface. The virtual assistant may include a speech-to-text component for receiving audio from an end user, interpreting speech content in the received audio, and converting the interpreted speech content to text. From an output perspective, the virtual assistant may include a text-to-speech component for receiving text to output as audio to the end user, converting the text to speech audio, and outputting the audio to the end user.

The message content may be a single message from the end user to the software service with an instruction to perform a function using content from the message. The message content may be a forwarded message or thread of messages to or from the end user from a third party (i.e. not the end user or the software service) such as a customer, potential customer, contact, etc. The message content may belong to a single session comprising a series of text inputs from the end user and responses from a virtual assistant, in a UI or webpage of the software service.

The message content is transformed into a search string at S102, for example to remove from a thread any message other than the most recent message or most recent exchange of messages, or to remove any messages preceding a particular event, such as the most recent execution of a processing function from the set of processing functions. The search string generated at S102 may be to generated by formatting the received message content as a text string, and may include speech-to-text conversion. A segmentation algorithm may be executed as part of the generating process at S102 to extract elements from the message in which a user intent may be expressed, for example, by identifying verb conjugations and segmenting to extract words associated with the verb conjugation.

The search string obtained at S102 by the software service is input to a searching mechanism at S103. The searching mechanism may be a full text search, may be vector-based, or may be another form of searching mechanism. The parsed search string is the received message, or an excerpt extracted therefrom, in a format appropriate for input to the searching mechanism at S103.

A search step is performed at S103 to extract the subset (which may be referred to as a shortlist) of processing functions from the full set of those available. Since the selection of a single processing function to execute is to be performed by a generic LLM that is an external service that has not been trained with knowledge specific to the domain of the software service, the search step at S103 serves to guardrail the selection of the LLM. The search step constrains the available choices of the LLM in order to suppress or remove altogether the risk of a hallucination type response by the LLM (that being a response which is apparently inconsistent with the logic of the prompt and/or ordinary principles of logic).

Optionally, the text descriptors of each member of the complete set are embedded into vectors to compose a searchable vector space of the text descriptors. The vector space is searchable at

S103, by the searching mechanism in response to the search string created at S102 being input. For example the received message content or an excerpt thereof is the search string. The search string may itself be embedded or vectorized by the searching mechanism as part of the functionality thereof. The searching step triggered by the input of the search string to the searching mechanism at S103 may be considered an initial filtering step, to filter out irrelevant candidates.

The searching at S103 is to return a subset of the complete set of available user content processing functions from which subset the LLM is asked to make a selection of a single subset best satisfying the user intent. Thus, methods including S102 leverage a first filtering technology, traditional vectorized searching, to reduce the number of options from the membership of the complete set to the membership of the subset identified in the search. Wherein the size of the subset is a configurable global parameter under control of the software service. To give a broad idea for the purposes of illustrating utility of the method, the complete set of user content processing functions may number of the order of 10 to 100 members, and the searching step S103 may be configured to return the three best matches. Rather than a fixed number of members of the subset, there may be a threshold minimum value of a similarity or other metric assessing likelihood of user intent matching the respective user content processing function.

In the case of vectorized searching, the searching mechanism may embed or vectorise the search string into a vector space. The process is equivalent (that is, using the same embedding or vectorizing parameters, embedding or vectorizing to the same vector space) to a process for embedding or vectorizing the text descriptors.

The searching mechanism ranks the text descriptors for semantic similarity to the search string. For example, in the vectorized searching example the metric for measuring semantic similarity may be vector distance from the vectorized or embedded search string to the respective vectorized or embedded text descriptor. A smaller vector distance indicates greater semantic similarity.

The searching mechanism and vectorized search space may be hosted by the software service provider (a native system) or may be a software function available as an on-demand service from a different provider.

The text descriptors of the top n ranked text descriptors (ranked for semantic similarity to the search string by the searching mechanism) are selected for inclusion in the LLM prompt generated at S104. The LLM prompt generated and output at S104 may be referred to as a processing function LLM prompt, reflecting the LLM prompt requesting return of an identity of a processing function to execute.

The LLM to which the processing function LLM is output (the first LLM) may be hosted externally and available to the software service as an on-demand service. Such an LLM may be generic insofar as it has not been fine-tuned or trained with training data specific to the software service and the functionality it provides. Alternatively, the LLM may be hosted by the software service (or may be stored externally but accessible only to the software service) and be at least partially trained or fine-tuned using training data including historical message conversations between end users and virtual assistant over the messaging interface, or other training data specific to the software service. For example, the said conversations are reviewed by a developer or a domain expert to provide a ground truth of the processing function that best matched the expression of user intent.

The number n is a positive integer equal to 2 or greater than 2. Examples include n=3, n=4, n=5. Advantageously, the size of the processing function LLM prompt is suppressed compared within including text descriptors of the full set of processing functions. Processing time by the LLM is reduced, data transmission bandwidth is reduced, and a cost incurred by submitting the processing function LLM prompt to the third party LLM provider is reduced (compared with an LLM prompt including the full set of text descriptors). By including the text descriptors from which the LLM is asked to select one best matching a user intent by the prompt, the LLM functions as an RAG AI model (retrieval augmented generative) AI. Risk of hallucinatory results from the LLM is avoided by the searching to reduce the full set of processing functions to the selected subset. Furthermore, by sending a prompt to a generic LLM the virtual assistant of the software service is able to interpret a broad spectrum of language usage without the need for domain specific training of a proprietary language model.

The message content, whether pre-processed, transformed, or otherwise modified at the software service, may be included in a processing function LLM prompt generated by the software service at S104. A procedure for transforming the message content is illustrated in Figure 1B and described in more detail below. Whether by the process of S201 to S203 of Figure

1B, or otherwise, an expression of user intent may be extracted from the message content itself or more generally from a messaging history between the end user and a virtual assistant. The same expression of user intent obtained at S203 for inclusion in the search string at S103 may be included in the processing function LLM prompt at S104. Thus, the processing function LLM prompt includes the expression of user intent, whether that be by direct inclusion of the message content or messaging history (including both sides of the conversation), or by some processing by the software service itself or an external LLM to extract or otherwise articulate an expression of user intent in the historical messaging between the end user and a virtual assistant underpinning the messaging interface.

The processing function LLM prompt further comprises a text descriptor of each member of a selected subset of processing functions, the subset being selected from a larger (i.e. having more members) set of processing functions. The set of processing functions may be all processing functions available to the software service in responding to messages received via the messaging interface.

The processing function LLM prompt requests that the LLM identify in the received message content an intended processing function from among the set of processing functions. Optionally, the processing function LLM prompt may also request that the LLM extract from the received message values for each of one or more data fields identified in the articulation of the selected processing function. The LLM is a generic generative AI model such as Claude or Claude Instant. The one or more data fields identified in the articulation may be values upon which execution of the function at S105 is dependent. For example, the message may be 'make me a report of the number of outstanding invoices issued to COMPANY A and total monetary value outstanding'. The processing function identified in the response at S104 is "get invoices", and the data field value extracted from the message is the identity of the particular customer or customer account. The get invoices function is dependent upon, or constrained by, the identification of the customer in the received message.

The processing function LLM prompt comprises an instruction, for example, 'based on the expression of user intent, identify which of the described/articulated processing functions the user wishes to have performed'. The LLM prompt may also comprise an instruction such as 'extract from the expression of user intent values for the data fields listed in the pertinent description/articulation'. That is, the processing function LLM prompt instructs the LLM to

perform at least one of two functions: user intent detection/identification, and data field value extraction.

The text descriptor may be referred to as an articulation, a digest, a representation, a description, a summary, or an abstract. The text descriptor may include one or more from among: a description of the content processing function, one or more keywords which (if identified in the message) indicate an intention to cause the content processing function to be executed, one or more data fields for which a value is required in order for the software service to perform the processing function, one or more historical received messages that were confirmed by an end user as expressing intent to have the respective processing function executed. Each member of the set of processing functions is a candidate for selection by the software service as the processing function best satisfying a direction, request, or intent expressed in the received message content. The description may include re-phrasings of any of the elements, to serve as context for the LLM in generating a response to the prompt, and also to improve search accuracy at S103.

The message content may comprise values of one or more parameters or data fields that are usable by the software service in performing a processing function on behalf of the end user. The parameter values or data field values may be extracted from message content or any other expression of user intent in the processing function LLM prompt by the LLM or by a component of the software service. The parameter values may constrain or otherwise guide the execution of the selected processing function at S105, as will be discussed in more detail below.

The processing function LLM prompt is submitted to an LLM such as Claude or Claude Instant, and at S104 a response obtained. Alternatively, another LLM such as ChatGPT, Lama2 etc, may be the recipient of the processing function LLM prompt. The format of the response may be predefined by the LLM or may be specified in the processing function LLM prompt. The response may be in a Json format. The LLM to which the prompt is output may be Claude 3 Sonnet; Claude 3 Haiku; Claude Instant 1.2; GCP Gemini pro; GCP Text Bison; GPT 3.5 Turbo; or GPT 4. The generating the prompt S104 may comprise selecting processing logic to execute in creating the prompt in dependence upon the LLM to which the prompt is to be output. The selection of LLM may be a global variable in the software system controlled by an administrator, or may be adaptive based on runtime parameters.

The LLM performs at least one of two functions in response to the processing function LLM prompt: user intent identification, and data field value extraction. Data fields are pieces of information required to fulfil the user intent. Wherein, the user intent is a processing function from among the subset of candidate content processing functions described/articulated in the LLM prompt. The response from the LLM comprises an indication of the intended processing function. The response from the LLM may also comprise the extracted values. The data fields are properties or parameters for which a value is required in order to perform the respective processing function. The processing functions are generic functions, such as create contact, create invoice, etc, which are tailored, personalized, or otherwise limited for the specific information contained within the message content, and which specific information may be extracted by the LLM responding to the processing function LLM prompt.

Optionally, the text descriptors may also include a list of parameters that the processing function requires to execute, and the prompt generated at S104 may also ask the LLM to extract, from the expression of user intent such as message history in the processing function LLM prompt, values of any of the listed parameters in the text descriptor of the processing function that the LLM identifies as best matching user intent.

The message content may identify a particular customer or customer account and request a report of number of outstanding invoices and total monetary value outstanding. The processing function is to “get invoices”, but the data field value extracted from the message content is the identity of the particular customer or customer account. In that way, the get invoices function is constrained by the identification of the customer.

A text descriptor or articulation may be composed of two elements. A first element may comprise one or more keywords, phrases, or other indicators that assist with identifying the pertinent content processing function as the user intent. For example, the first element may be an explicit command in a message forwarded to the software service by the end user, such as ‘create quote’, ‘create invoice’, or ‘update contact’, or may be the equivalent in a message to the end user from a contact of the end user (and which is forwarded or otherwise relayed to the software service by the end user). A second element may comprise one or more data fields required to execute the pertinent content processing function. It is noted that, for each element, the prompt includes instructions to guide the LLM as to the usage of the element. The LLM prompt asks the

LLM to extract a user intent, and to extract data field values for performing or executing the content processing function.

The processing function LLM prompt contains all of the information that the LLM requires to perform the requested function or functions. So, the LLM identifies user intent as one of the processing functions from among the subset described in the prompt itself, by finding the text descriptor from among the text descriptors of the subset of processing functions ranked in the top n for semantic similarity by the searching mechanism at S103. The LLM is a generic third-party on-demand service and is not specifically pre-trained in the domain of the processing functions. Advantageously, one or more from among edits, deletions, and modifications, can be made to the complete set of articulations without needing to re-train any AI model. The processing function LLM prompt further includes the received message content from the end user, or a representation thereof, or an excerpt thereof, along with additional messaging content such as further historical messages received from the end user, and messages output by a virtual assistant to the end user in a conversation conducted via the messaging interface.

The description or articulation of each of the content processing functions may distinguish between required and optional data field values. For example, in the case of creating a draft invoice document, if a line item price or cost is specified in the message content then that is extracted by the LLM, but it is not a requirement. The create invoice API to which the JSON generated by the LLM is passed may be configured to use a default line item price where none is specified in the JSON input. Required and optional data fields may serve two purposes. Firstly, they provide context for the LLM: if there are four required data fields in a particular articulation, and the LLM identifies a value for each of the required data fields in the message content, then that may help the LLM to determine that the user intended processing function is the processing function mapped to the particular articulation. Furthermore, in case the processing function LLM prompt requests that the LLM perform data extraction, the required and optional data fields act as instructions for the data extraction. Extracted data field (parameter) values may be used by the software service to determine which of plural sequences of actions to follow in the execution of the selected processing function at S105, as set out in more detail in Figures 2 to 5D and the associated description.

The processing function LLM prompt comprises a text descriptor of each of a subset of processing functions. The processing function LLM prompt asks the LLM to classify a user query

(i.e. the received message content) in terms of one of the subset of processing functions. The processing function LLM prompt may also ask the LLM to identify and extract any parameters (data field values) that have been submitted by the end user as part of the user query, and optionally also to generate a response saying what action it intends to take and whether any further parameters are required.

The software service may be an SaaS architecture based software service in which content processing functions are executed via one or a series of RESTful API calls. The software service stores a mapping of processing functions to API calls, so that upon selection of a processing function by the LLM and return of said identification by the LLM at S104, the software service, based on the mapping, causes the selected processing function to execute by a procedure such as illustrated in Figures 2 to 5D.

At S105 the software service causes the selected processing function to be executed. The processing function may be performed in accordance with one or more data field or parameter values extracted from the message content by the LLM. The software service may itself perform the processing function, or may trigger the processing function. The software service may forward the Json payload (that is, Json content) itself to the selected processing function via an API, or may extract data from the Json payload for submission to the selected processing function via an API. Thus, the selected processing function is caused to be performed. The processing function may be to create or update a contact entry in a contacts database, or to create or update a document such as an invoice or a purchase order. In case the processing function relates to generation or update of a document for the end user, the generated document may be transmitted to the end user for use or presented to the end user within an interface of the software service for approval. The end user may approve the generated document by logging into the software service.

Figures 2 to 5D and the associated description disclose a mechanism for executing the selected processing function, based on traversal of a data graph representing the selected processing function.

The processing function may comprise generating content comprising a new or updated document or a new or updated database entry, or a report or other information from content specific to the end user stored by the software service. Executing the processing function may

comprise documentation or data generated or updated in the course of the execution being at least one of: transmitted to the end user via the messaging interface; presented to the end user via a graphical user interface; or stored at a storage location accessible to the end user.

The set of processing functions may include processing functions which: generate, create, update, modify, report on, manipulate, or delete, documents and data (i.e. content) specific to the end user (or to an organization to which the end user belongs). For example, the processing functions may include a processing function that assimilates end user documents and data stored by the software service to create a report. The processing functions may include a processing function that creates or updates documents or database entries stored by the software service in a storage area accessible to the end user and specific to the end user or their organization.

The processing functions may require data field values extracted from the received message content to be matched to existing data field values stored in a database at the software service. For example, to match an entity identified in the message content with an entity existing in a data source stored by the software service, such as a database. The data source being data stored at the software service relating to the end user and its organization. Causing the selected processing function to execute in accordance with the extracted data field values may include calling a data source of the software service in real time to verify one or more of the data field values extracted from the message content.

Examples of processing functions include:

- Creating a new contact entry in a database of contact entries specific to the end user, the database of contact entries being a component of the software service. In which case the new contact entry may comprise one or more of the data field values extracted from the received message content. For example, a unique identifier of the contact entry such as contact name may be compared with a list of contact names stored in a contacts database by the software service in a contacts database stored on behalf of, or otherwise accessible to (for example by being stored on behalf of an organization to which the end user belongs) the end user.
- Updating an existing contact entry in a database of contact entries specific to the end user, the database of contact entries being a component of the software service. The

updated contact entry may comprise one or more of the data field values extracted from the received message content.

- Creating a new invoice document defining one or more line items and an associated cost per line item. The new invoice document may comprise one or more of the data field values extracted from the received message content. For example, a unique identifier of the line item such as a line item name or line item label, descriptor, or number, may be compared with a list of line items stored in an invoicing database by the software service in an invoicing database stored on behalf of, or otherwise accessible to (for example by being stored on behalf of an organization to which the end user belongs) the end user.
- Updating an existing invoice document. The updated invoice document may comprise one or more of the data field values extracted from the received message content.
- Create a report or information based on processing, extracting information from, or assimilating, end user content stored by the software service.

End user content may comprise documents and data stored by the software service on behalf of the end user.

Once the processing function has been performed at S105, a result may be output to the LLM in a further prompt also comprising the expression of user intent and requesting that the LLM respond with a reply to a most recently received message based on, including, or directing the end user to, the result. The reply generated by the LLM is output to the end user via the messaging interface.

Alternatively, there may be no results output, simply a text string indicating that the processing function has been performed. Or the processing functions may themselves include output steps.

The LLM in question may be a generically pre-trained off-the shelf generative AI algorithm that requires no training specific to the method of Figure 1A. That is, it requires no prior knowledge at the LLM of the software service or the set of processing functions (because they are described/articulated in the prompt). The computer-implemented method of Figure 1A may be performed with no training phase to customize the LLM (nor any other AI algorithm) for the particular implementation scenario. The LLM may be provided by a third party (i.e. a party that is

not the end user or the software service provider) that provides a generic generative LLM as an on-demand service. Advantageously, the LLM provider may be selected based on ability to interpret a broad input vocabulary, so that constraints associated with properly training a proprietary AI model with domain-specific data are avoided.

The software service provider provides end users with access to a set of processing functions simply by sending or forwarding a message to the software service. The message may be unstructured text and need not expressly specify which function is to be performed, as long as the function is resolvable from the other functions based on the message content. Generative AI, in the form of the LLM, surfaces user intent and may also perform data extraction from the message content (which is forwarded to the LLM in the prompt) without undergoing any sort of pre-training specific to the task. Therefore, no training dataset is required.

In the computer-implemented method of Figure 1A, once the end user has forwarded the message content at S101, the process is automatic.

In the computer-implemented methods the messaging interface may be a natural language interface so that unstructured text may be submitted to the messaging interface by the end user. The process is triggered by receipt of a natural language message from the end user at the messaging interface. The natural language messaging interface may be a chatbot, AI assistant, or virtual assistant.

Figure 1B illustrates a computer-implemented method. The method of Figure 1B incorporates the method of Figure 1A and further includes steps S201 to S203, which define a procedure for transforming, enhancing, or augmenting, the search string.

In the methods of Figures 1A and 1B, the messaging interface may be for exchanging messages with a virtual assistant. The messaging interface may be considered to be a component of the virtual assistant, or the virtual assistant may be classified as the processing logic underpinning the processing of messages received at the messaging interface. Messages exchanged (both ways) in a single session may be referred to as a conversation.

At S201 a further LLM prompt is generated, which may be referred to as a search string LLM prompt. The search string LLM prompt comprises message content including at least a most

recent message received from the end user, and optionally all messages received in the conversation, or all messages received and transmitted in the conversation (i.e. includes messages output by the virtual assistant and received from the end user). Optionally, a bar is applied to how far back in the conversation message content is included in the search string LLM prompt. For example, a last predefined number of messages, all messages received within a predefined time duration, or all messages received since a predefined event, such as since a last processing function execution prompted by the conversation.

The search string LLM prompt is to engage an LLM in automating the creation of a search string that represents an expression of user intent, rather than relying on the content of a most recently received message. For example, if the most recently received message was 'yes', then inclusion in the search string LLM prompt of both sides of the conversation provides visibility of a question to which the 'yes' was an answer. The search string LLM prompt including historical messages in the conversation enables the LLM to backtrack in the conversation and to better understand where user intent is expressed.

The search string LLM prompt may also prompt the LLM to augment the expression of user intent with contextual information for interpreting the expression of user intent. Examples include business type, name, territory. Any information that may be of use.

At S202 the search string LLM prompt is output to an LLM. The LLM to which the prompt generated at S201 is output (the second LLM) may be the same LLM to which the prompt generated at S104 is output. The LLM to which the prompt generated at S201 is output may be different to that which the prompt generated at S104 is output.

The LLM to which the search string LLM is output (the second LLM) may be hosted externally and available to the software service as an on-demand service. Such an LLM may be generic insofar as it has not been fine-tuned or trained with training data specific to the software service and the functionality it provides. Alternatively, the LLM may be hosted by the software service (or may be stored externally but accessible only to the software service) and be at least partially trained or fine-tuned using training data including historical message conversations between end users and virtual assistant over the messaging interface, or other training data specific to the software service. For example, it might be that the said conversations are reviewed by a developer or a

domain expert to provide a ground truth of an optimal search string to extract from a messaging history.

The search string LLM prompt is submitted to an LLM such as Claude or Claude Instant, and at S203 a response obtained. Alternatively, another LLM such as ChatGPT, Lama2 etc, may be the recipient of the search string LLM prompt. The format of the response may be predefined by the LLM or may be specified in the search string LLM prompt. The response may be in a Json format. The LLM to which the prompt is output may be Claude 3 Sonnet; Claude 3 Haiku; Claude Instant 1.2; GCP Gemini pro; GCP Text Bison; GPT 3.5 Turbo; or GPT 4. The generating the prompt S201 may comprise selecting processing logic to execute in creating the prompt in dependence upon the LLM to which the prompt is to be output. The selection of LLM may be a global variable in the software system controlled by an administrator, or may be adaptive based on runtime parameters.

At S203 the response from the LLM is received and parsed into a search string for use at S103. Steps S103 to S105 are as discussed in relation to Figure 1A.

An exemplary data graph is illustrated in Figure 2. The data graph is a directed data graph representing the processing function by interconnected nodes 202, each node 202 representing one of the component actions of the processing function. Directed edges 204 define adjacency between component actions in the different sequences by which the processing function is executable.

Causing the selected processing function to execute may comprise performing the component actions represented by graph nodes on the execution traversal path in sequence, starting at a starting graph node for the processing function and determining the execution traversal path during the execution by, when the most recently performed component action is represented by a graph node from which two or more directed edges originate, selecting which of the two or more directed edges to follow in dependence upon an outcome of a component action already performed in the execution.

The data graph may be a directed data graph representing the processing function, each of the component actions being represented by a graph node, and each of the plurality of different executable sequences of component actions being a represented by a traversal path connecting

graph nodes representing component actions in the executable sequence between a traversal path end node and a traversal path start node.

Causing the selected processing function to execute may comprise performing the component actions represented by graph nodes on the execution traversal path between the traversal path end node and the traversal path start node in forward or reverse sequence, starting at either the traversal path start node or the traversal path end node, and determining the execution traversal path during the execution by, when a plurality of valid next nodes in the execution traversal path are available, selecting which of the next nodes to include in the execution traversal path in dependence upon an outcome of a component action already performed in the execution traversal path or in dependence upon information extracted from the received message content.

The node may itself store the processing logic required to execute the represented action, or may store a label mapping to the said processing logic, or a pointer to the said processing logic (which mapping or pointer the controller is configured to follow).

The methods of the Figures may be performed by a controller, being a software module or software component of the software service providing a virtual assistant to end users. The virtual assistant underpins or includes a user interface such as a messaging interface to output and receive messages to/from an end user. The methods of the figures form part of the server-side functionality underpinning the virtual assistant. The controller interfaces with other functions of the software system, for example via API calls, to cause the actions represented by the graph nodes to be performed, and thus the processing function to be executed. The controller uses the data graph as an encoded set of rules determining how to execute the processing function.

The execution traversal path is not predetermined. It is determined during execution according to data retrieved and generated in the execution. The execution traversal path is not unconstrained: there are a finite number of different sequences of component actions, defined in the data graph by directed edges between nodes representing component actions. However, some processing functions may be executable by following a loop which is repeatable, that is, a directed edge followed in an execution traversal path may return to a node that has already been included on the same execution traversal path. The execution traversal path may be determined in a forward direction by following outgoing nodes from the execution traversal path start node, or may be determined in a reverse direction by following incoming nodes to the execution

traversal path end node. Noting that the end node is predetermined by the selection of the processing function, since each processing function maps to a specific end node.

The node at which to start the execution traversal path may be predetermined for the processing function, i.e. so that selection of processing function at S120 determines what the starting node will be. Alternatively, different user intents in the received message may map to different starting nodes within the same data graph. Or, different data fields instantiated by information extracted from the received message may map to different starting nodes. Or, different data field values for a particular data field may map to different starting nodes. In summary, the starting node may be predetermined or may be adaptively determined.

The data graph of Figure 2 shows that each of the data graph nodes maps to a component action of the processing function, as indicated by the labels f1 to f8. In a particular example:

f1 = get_invoices (API call)

f2 = generate payload (with late fee payment)

f3 = update_invoices (API call)

f4 = return invoice number, URL

f5 = extract line items from invoice

f6 = get_org_shortcode_url (API call)

f7 = return contact name, URL

f8 = return line items, unpaid amount

Different user intents map to different end nodes. For example, if an extracted user intent from the received message indicates that the end user wishes to add a late payment fee to an invoice, the end node is f4 and the sequence of component actions may be f1 – f2 – f3 – f4. If the extracted user intent from the received message indicates that the end user wishes to fetch unpaid invoice details, the end node is f8 and the sequence of component actions may be f1 – f2 – f5 – f5 – f4 – f7 – f8.

Figure 3 illustrates a data graph representation of a processing function. The boxes with square corners represent data artefacts or data field values, and the boxed with rounded corners represent processing tasks. Figure 3 is described in the context of a forward traversal path determination. It may be that the selection of the processing function maps to a single end node and a single start node, so that the determination is to select which valid path to follow between the start node and end node mapped to be the selected processing function. Figure 3 illustrates

a data graph usable by a controller for determining how to execute a processing function, such as adding a late fee to an invoice. In the example of Figure 3, a starting node 302 represents a message exchange with an end user. The content of the message exchange is dependent upon a state of the execution. For example, at a first performance of action 302, the user may be asked for more information about the late fee they wish to add. The nodes 304 represent data artefacts, being documents or data field values. In dependence upon a type or class of data artefact obtained in an exchange of messages with a user, an outgoing edge from starting node 302 to be followed in the execution traversal path is selected by the controller.

Figure 3 illustrates a data graph in which data artefacts are represented by graph nodes. In Figure 2, the data artefacts are not represented by graph nodes. Methods such as illustrated in the Figures may operate with either data graph configuration.

Examples of data artefacts that may be extracted from the received message at 302 include a name, a date, an overdue flag, an amount or order flag, a date range, an amount, or a percent amount. An outgoing edge from each of the data artefact nodes 304a connects to a corresponding component action node 306a. For example, if the extracted data artefact is a name, the corresponding component action is to search an invoice data repository using name as a search field. If the extracted data artefact is a date, the corresponding component action is to search an invoice data repository using date as a search field. If the extracted data artefact is an overdue flag, the corresponding component action is to search an invoice data repository using presence/absence of overdue flag as a search field. If the extracted data artefact is an amount or order flag, the corresponding component action is to search an invoice data repository using amount or presence/absence of order flag as a search field. Or the extracted data artefact may be a date range of late fees already levied, for which the corresponding action is search late fee by history. The extracted data artefact may be an amount, in which case a flat fee line item is created, or the extracted data artefact may be a percentage, in which case a percentage fee line item is created.

Data artefacts resulting from the component actions 306a are represented by data artefact nodes 304b linked to the corresponding component actions by an outgoing edge from the component action node 306a to the corresponding data artefact node 304b. The data artefacts represented by data artefact nodes 304b are either a list of invoices (according to the information

contained in the data artefact at 304a in the same traversal path), or a set of late fee options (also according to the information contained in the data artefact at 304a in the same traversal path).

A next component action 306b is, in sequences in which the data artefact is a list of invoices, is a find an invoices component action 306b. The controller causes the find an invoices component action to execute and provides as inputs to the action the outputs of at least one of the earlier actions in the execution traversal path ('ask user' and, for example, 'search invoice by amount'). The find an invoice action at 306b enables the controller to loop back to the ask user action 302 to resolve between multiple returns in the list of invoices, or to provide further information if no invoice is returned. The ask user action 302 may include generating an LLM prompt including a description of a state of the execution, for example, 'list of invoices has returned multiple invoices where only a single invoice is required, construct a message to the user asking the user to resolve between the invoices in the list <include list>'. The prompt may further include the history of the messages exchanged with the user so far in the execution. The controller outputs via the messaging interface the message from the LLM, and receives a reply. Implicit in the ask user action 302 is an information extraction step, which may be performed by the controller itself, or by the controller sending a prompt to an LLM requesting that the LLM extract a data field value from the reply. The controller then proceeds with the data artefacts already obtained in the execution and the new data field value, and upon reaching component action 'find an invoice' again at 306b, executing the action results in a single invoice ID represented by graph node 304c being output. A corresponding 'find a late fee' component action is to identify a specific late fee to add to an invoice, wherein multiple valid options in the data artefact 304b obtained at action 306a cause the controller to follow an outgoing edge looping back to the 'ask user' action 302. As described in relation to the 'find an invoice' action, the 'ask user' action is dependent upon a state of the execution, which dependence may be manifest by including a description of the state in an LLM prompt requesting the LLM to generate a message to output to the end user to resolve between the multiple valid late fee options.

The controller may also direct the graph traversal path to the ask user action in case clarification is sought on a question asked by the user, or an intent expressed by the user. Such a scenario may be considered a suspension of the execution graph traversal path, causing an edge to be followed to an 'ask user' action such as ask user action 302. In a particular example, the LLM prompt requesting that the LLM extract user intent in the initial selection of a processing function, or the LLM prompt requesting that the LLM extract information, may also request that the LLM

indicate a level of confidence in the extracted information or user intent, as appropriate. In case the indicated confidence is less than a predefined threshold, the controller may revert the execution traversal path to the ask user action 302, which would cause an LLM prompt to be generated and output including execution state, which would be insufficient confidence level to proceed, and message history. The LLM would then generate a clarifying question for output to the user at 302 to prompt a user response which would add confidence to the extracted information or user intent, and allow the execution traversal to proceed beyond the stage at which it was suspended. Such a procedure allows for the resolution of informative errors that may otherwise be caused by proceeding with the execution traversal path based on information in which there is a low confidence level.

A component action at 306c is to add a calculated late fee amount to an identified invoice. If either data artefact (invoice identifier or late fee amount) is absent, the controller follows an outgoing edge returning to the ask user action 302 and, as above, an LLM prompt containing a description of the state of the execution, a messaging history or excerpt thereof, and a request to respond with a message to output to a user. In the present case, if the invoice is obtained, the user may be asked to provide information about the late fee to be added, and the controller determines how the execution traversal path should proceed based on the response, from among the available options indicated by outgoing edges from the node 302.

Once the execution traversal path reaches action node 306c, and both required data artefacts have been obtained by the controller in the course of executing the current execution traversal path, the action of adding the late fee to the invoice is performed. Optionally, the action, or any action on an execution traversal path, may be initially performed and assigned a 'pending confirmation' status by the controller, wherein dependence upon a user reply to a confirmatory message, the action may be reversed (in the case of a negative response) or confirmed and committed (in the case of a positive response). In the example of the late fee being added to the invoice, an output message at S308 may be generated by an LLM responding to a prompt generated by the controller and including either a data artefact updated or created by the pending confirmation action, a link to the said data artefact, or a description of the said data artefact or the change/update.

Figure 4 illustrates a computer-implemented method. At S401 a full set of processing functions and their respective knowledge graph (wherein knowledge graph is an example of data graph)

representations are obtained. S401 may include a knowledge graph building process, which may be manual or automated. Each knowledge graph represents a respective processing function that can be executed to fulfil an end user request in the software service. The component actions and the paths between them are encoded in the graph. Directed edges between component actions indicate where the output of one component action (the directed edge originates at a graph node represent the one component action) is usable as the input to another component action (the directed edge terminates at the graph node representing the another component action).

A text descriptor of the data graph representation of the processing function may be automatically extracted from the data graph representation. Alternatively, the data graph representation may be extracted from a text descriptor of the processing function. The text descriptor of the processing function is includable in an LLM prompt at S403, and as set out in more detail below in the description relating to Figures 1A and 1B. The text descriptor is searchable by a searching mechanism at S402, and as set out in more detail below in the description relating to Figures 1A and 1B.

The data graph representations and tool descriptors are stored and are accessible to the searching mechanism and the controller, wherein controller refers to the software service components underpinning the software service's responses to the end user queries received via the messaging interface, such as the virtual assistant.

Receipt of end user message content 402 triggers a tool retrieval step. The function retrieval step S402 may be performed by inputting the end user message, or excerpts thereof, to a searching mechanism as a search string, as set out in more detail below in the description relating to Figures 1A and 1B. The function retrieval step S402 returns a subset or shortlist of processing functions from the full set of available processing functions. The tool retrieval step S402 extracts a shortlist of potentially relevant functions from the full set according to the received message. The tool retrieval step S402 may also extract one or more pieces of information from a messaging history between the user and the virtual assistant, with which to constrain or otherwise guide or inform the selection of the shortlist from among the full set. The size of the shortlist may be, for example, three, four, five, six, or seven.

At S403 an LLM prompt is created including the received message 402 or an excerpt therefrom. The LLM prompt also includes the text descriptors of the shortlist selected in the retrieval step S402. The LLM prompt requests that an LLM chooses the single processing function from the shortlist that best matches a user intent expressed in the received message 402. The LLM may reply either with an identification of a single processing function most relevant to the received message 402, or none.

At S404 an optional validation step is performed. The optional validation step S404 may be used when there is an existing processing function being performed and so there is a pending execution status, being a particular data graph node that represents a current node reached in an execution traversal path being performed. If the selected processing function from S403 is consistent with traversal paths continuing from the current node, then the execution traversal path being performed continues accordingly. Otherwise, either the current execution traversal path is cancelled, or, the selected processing function from S403 is added to a stack as a next function to execute following completion of a current execution.

At S405 the data graph traversal from a starting node corresponding to the selected processing function from S403 begins. The purpose of the traversal is to fulfil the required properties for a data artefact creation or update action in the processing function, or to fulfil some other completion condition. All possible sequences of component actions to reach the completion condition are represented by the arrangement of nodes and directed edges in the data graph representation of the processing function. A controller at the software service coordinating processing responses to virtual assistant messages determines an execution traversal path, being one among all plausible traversal paths in the graph, in dependence upon data artefacts and data values obtained in the execution, and on outcomes of preceding actions in the execution. More detail on graph traversal is set out elsewhere in the disclosure in Figures 2, 3, and 5A to 5D.

The data graph traversal S405 may comprise one or more of steps S406 to S408, which may be performed in any combination, any number of times, in combination or in isolation, according to the progress of the traversal. S406 is property extraction, which may comprise extracting one or more data field values from either the received message, or data artefacts obtained in preceding component actions in the execution. Property extraction may be performed using processing logic forming part of the software service and accessible to the controller of the response to the

received message via an API call or equivalent. In some cases, property extraction may be performed by sending a prompt to an LLM comprising a list of data fields for which values may be present in a data item, the data item, and a request to extract values of listed data fields where values are present in the data item. The extracting properties may be performed at a component action represented by a node on the execution traversal path, and the extracted properties used to determine which of plural outgoing edges to follow from the node. It is noted that the determination may be based on the data field that is instantiated by the value, rather then the value itself. For example, if a value of a 'name' data field is extracted, the execution traversal path may follow a different directed edge than would be followed if a value of a 'date range' data field is extracted. In addition to being listed in the prompt, each data field may be described. The listing, and where included the descriptions, may form part of the text descriptors of the respective processing functions.

The compose execution path step S407 may follow from an extract properties step, or may otherwise form part of the graph traversal process S405, for example, whenever a graph node is encountered from which plural directed edges originate. Optionally, each directed edge is labelled with a logical statement enabling the controller to determine in which circumstances the directed edge should be followed. For example, 'if >2 results returned', or 'if name field instantiated'. Alternatively, the inclusion of data artefact nodes in the data graph may serve an equivalent purpose, as described above in relation to Figure 3.

The generate dialogue step S408 may include creating a prompt to send to an LLM describing a current graph state (that is, a current node in the execution traversal path) and requesting that the LLM, for example, generate a message to output to the user describing the current state, or requesting information on how to proceed, for example, where multiple valid options exist. Steps S406, S407, and S408 form part of the data graph traversal step S405, as described above in relation to Figures 2 to 3.

Figure 5A illustrates a worked example of a computer-implemented method. The four software components are elements of the software service providing data processing and storage functions to users via the internet. A chat interface 501 is an example of a messaging interface, and is an end user interface component of a virtual assistant function from which a user can trigger processing functions. The processing functions may be available to the end user via other mechanisms, such as by navigating a menu or series of menus, search fields, etc, but the virtual

assistant provides a more direct and convenient processing function triggering mechanism. The chat interface 501 is an end user facing component of a virtual assistant. At a server side the virtual assistant comprises a controller which may comprise a service 503, a model 505, and a graph stack 507. The service performs the execution of processing functions, and in particular, the sequence of component actions forming a processing function. The model 505 comprises at least processing logic for interfacing with an external LLM, and may also comprise the LLM. The model 505 may be operable to generate LLM prompts, and to receive replies from the LLM and parse and distribute the replies to the stack 507, the service 503, and the chat interface 501.

A message 502 from an end user is received by the chat interface 501. In the worked example of Figure 5A, the message is “add $50 late fee to John’s invoice”. In a step not shown in Figure 5A, the service 503 may parse the received message and execute a search against a search space composed of text descriptors of a complete set of processing functions, and based on the ranked results, select a top n-ranked processing functions as a shortlist or subset from which to ask an LLM to select a single processing function to perform. Alternatively the LLM may be requested to select from the complete set of processing functions.

The virtual assistant creates an LLM prompt 504 comprising text descriptors or articulations of either the full set of processing functions or a subset thereof, the received message, and a request that the LLM select a processing function best matching user intent expressed in the received message. The processing functions may be referred to as tools. The LLM prompt 504 is output to the LLM via an LLM call, and in response a selection of a single processing function received. In the worked example, the selection is the ‘Add Late Fee Flat Fee’ processing function. The LLM prompt 504 is an example of S403 of Figure 4.

The stack 507 is operable to determine which sequence of actions to perform in executing the processing function, exemplary of step S405 of Figure 4. Plural different sequences of actions are permissible, as encoded in a data graph (which may be referred to as a knowledge graph). The stack 507 comprises processing logic for retrieving the graph representing all permissible sequences (represented by traversal paths) of component actions that may be performed in executing the processing function (i.e. a generic data graph representation of the processing function) the selected processing function, and for pruning and updating a working copy to determine a single execution traversal path to follow in the specific set of circumstances of the current execution.

A data graph lookup 506 is executed to load an accessible of a data graph representing the selected processing function to the graph stack 507 from which an execution traversal path is determined according to data retrieved from user messages and by performing actions on the execution traversal path.

A data extraction step 510 is performed by the model 505 by creating an LLM prompt to the LLM comprising the initially received message 502 and a list of data fields to instantiate. The list is generic for the selected processing function, and may be stored in one or both of the data graph representation of the selected processing function and the text descriptor of the selected processing function. The LLM prompt requests that the LLM extract from the message 502 values of any of the listed data fields for which values are provided in the message 502. In the worked example, the values are name="John", which is on a traversal path corresponding to a 'search invoice by name' component action, and amount="50", which is on a traversal path corresponding to an 'add late fee flat fee' component action.

The model 505 feeds the obtained data field values to the graph stack 507, which values prune the data graph 512 (i.e. because traversal paths from a current node which leverage data fields for which values are not obtained are pruned, that is, excluded from inclusion non the execution traversal path).

Pruning is an example of a process of determining which of plural outgoing edges to follow by a process of elimination based on logic associated with the outgoing edges that is assessed by the virtual assistant. In ruling out an outgoing edge (for example, because it requires a data field value that the virtual assistant has not obtained in the session), the outgoing edge is 'pruned' from the permissible options.

The virtual assistant determines that there is only one permissible outgoing edge in the pruned data graph 512 to follow based on the obtained data field values, and follows the outgoing edge to a node representing the 'search invoice by name' API call. The graph stack 507 notifies the model 505 that the next component action is 'search invoice by name', and an API call 514 corresponding to the node is performed by the virtual assistant. In the example of Figure 5A, the API call is to an API for searching invoices using a value of the name data field.

A response 516 to the API call is received by the virtual assistant. In the example of Figure 5A the response 516 comprises multiple returns. There are plural invoices for customers named "John". The virtual assistant updates 518 the execution traversal path 520 to determine how to proceed. That is, the outcome of the component action at the current node is that there are multiple returns, and an outgoing edge to follow from the current node is selected accordingly.

The data graph comprises an outgoing edge to a node representing an LLM call action to generate a natural language message to the user requesting the user to select from a list. An LLM prompt 522 is created and output to the LLM. The LLM prompt notifies the LLM of the execution state, for example, in the adding a late fee to an invoice processing function, multiple invoices have been returned, and that a selection of single invoice by user required, and requests that the LLM compose a natural language message 524 to output to the end user via the chat interface.

In the worked example of Figure 5A, the natural language message being output is "John has two outstanding invoices: -Inv1, overdue by 5 days; -Inv2, due in 2 days. To which invoice do you want to add the late fee?".

Figure 5B illustrates a continuation of the computer-implemented method of Figure 5A.

A message 526 is received via the chat interface 501. The message "tell me about the overdue one" is a digression in the context of the execution traversal path. It does not express a user intention consistent with the execution traversal path currently being followed in the graph stack, nor does it provide information relating to a data field value being resolved by the execution traversal path. The tool classifier LLM call at 528 (as described above at LLM call 504) indicates the user intent aligns with a processing function to get invoice details. The selected processing function, or tool, is mapped to a data graph from among a set of data graphs mapping to the set of processing functions.

Since the pending execution traversal path 532 is for a different processing function, the retrieval of the get invoice details data graph 530 and transfer to the stack 507 effectively pushes the new processing function to the front of the stack and hence execution of component actions in the execution traversal path 532 is suspended until completion of execution of the new processing function (which will be represented by a new execution traversal path 536).

The new data graph is initiated by a populate data fields data extraction action 538 equivalent to populate data fields 510. In particular, rather than including solely the received message 526 in the prompt, the messaging history including messages 502 and 524 is included, so that the LLM determines that the invoice number data field can be instantiated with the value inv1. The new data graph is pruned per the outcome of the data extraction action 538 to obtain a pruned data graph 540 in the stack. In particular, only one outgoing edge from a starting node corresponds to an instance of the invoice number data field, so that outgoing edge is followed by the virtual assistant.

A next node on the execution traversal path of the new processing function represents an API call 542 to an invoice details retrieval API specifying that the invoice for which details are requested is associated with invoice number inv1 per the data extraction 538. The virtual assistant, via the API call 542, obtains 544 invoice details for the specified invoice. The virtual assistant resolves 546 the node in the execution traversal path which corresponds to a completion condition for the data graph representing the new processing function, and the graph stack is updated by the removal 542 of the completed data graph. The suspended execution traversal path is effectively returned 544 to a currently executing state from the suspended state.

The completion condition for the get invoice details processing function also has an associated component action, which is a success dialogue LLM call 548. The virtual assistant creates an LLM prompt describing a current state of the new execution traversal path and the suspended execution traversal path (invoice details obtained for specified invoice, previous traversal path suspended at user choosing invoice from list), associated details (outcome of the component actions performed in the execution traversal path for the function), and the messaging history, and asks the LLM to compose a natural language reply to the message 526 using the details obtained at 544. The reply is output 550 to the end user via the chat interface 501. In the worked example of Figure 5B, the reply is 'The overdue one was to XYZ Corp for $60 of electrical supplies. To which invoice do you want to add the late fee?'.

The computer-implemented method continues at Figure 5C. A resolving message 552 is input to the chat interface by the user. As in steps 540 and 528, the received message 552 is included in an LLM prompt along with a description of the current state of the execution traversal path at the top of the graph stack 507, which is the traversal path that was suspended by the digression message 526. The present state informs the LLM that the selection of an invoice is required, and

the LLM prompt responds to the virtual assistant notifying the virtual assistant that the message 552 is selecting an invoice. A populate data fields step 564 equivalent to step 538 above returns the invoice number inv1, which is added to existing instantiation of the amount data field at $50. The suspended execution path 558 is pruned accordingly. In other words, an outgoing edge from a current graph node is selected according to the content of the received message 552. Since the add late fee processing function requires an invoice and a late fee amount in order to complete its function, it now has values of all required data fields. An optional approval step is included in the data graph of the add late fee processing function, which is represented by a next node on the execution traversal path 564. The virtual assistant therefore must execute an approval response action 566, which includes creating a prompt to the LLM with a current state (message history, data field values obtained for invoice number and late fee, specification of the values, approval required) requesting a natural language message to output to the user in reply to message 552. The message 570 is obtained by the virtual assistant from the LLM call 568 and output to the user via the chat interface 501. In the worked example, the message 570 is "Ok, we can add a $50 late fee to Inv1. Do you want to go ahead?".

The message 570 is an example of seeking a positive or negative reply from the end user. The purpose is to verify, or to validate, information extracted in the execution of one of the component actions. A positive reply indicates approval and the virtual assistant may proceed with the graph traversal. A negative reply, on the other hand, indicates disapproval, and the virtual assistant may repeat the information extraction with different parameters, or to extract replacement information from a message input to the user interface from the end user. The different parameters may be obtained automatically or may be obtained by reverting to the end user.

The computer-implemented method continues at Figure 5D. A confirmatory message 572 is input to the chat interface by the user. As in steps 574, 540 and 528, the received message 572 is included in an LLM prompt along with a description of the current state of the execution traversal path at the top of the graph stack 507, which is the instantiation of the required data fields and the seeking approval state. The present state informs the LLM that an indication of approval or disapproval is required. The LLM informs the virtual assistant that an indication of approval is present in the message and at 576 the virtual assistant matches the information to the current execution traversal path 578. Pruning of the data graph based on a positive response results in a pruned data graph 580. The virtual assistant updates the data fields with the approval value so that data fields invoice number, amount, and approval, are now instantiated. Since the

add late fee processing function requires an invoice and a late fee amount, and an approval, in order to complete its function, it now has approved values of the required data fields. The outgoing edges from the node representing seek approval from user correspond to positive and negative response. In the present case, the response was positive (interpreted by LLM in response to prompt 574) and so the execution traversal path proceeds along the corresponding outgoing edge to a graph node representing an update invoice API call 584. The API call 584 is made with the late fee value and the invoice number, which causes update of the invoice data artefact (underlying data and/or document), and in response an indication 585 that the update was successful. The virtual assistant obtains payload details (the updated invoice) and resolves the execution traversal path 586. In the context of the graph stack 507, the execution traversal path has reached a completion condition 587. Reaching a completion condition triggers a success dialogue LLM call 588, which creates and outputs to an LLM a prompt comprising the payload and a description of a state of the execution traversal path (completion condition met, details of updated invoice, message history) requesting a natural language message to output to the user in reply to message 572. The message 589 is obtained by the virtual assistant from the LLM call 588 and output to the user via the chat interface 501. In the worked example, the message 589 is "Great, we've taken care of it. Do you want to send the updated invoice?".

Intent Classification: Summary of Chat History

LLM prompts requesting the LLM determine or classify user intent expressed in a message (such as at S104, 504, S803) may be composed of at least a most recent message received via the chat or messaging interface from the end user, and may include historical messages exchanged with the software service via the interface in the same session.

Optionally, the historical message exchanges (which may be referred to as session history, chat history, or messaging history) may be submitted to the LLM with a request to return a summary. The summary may then be included in the new prompt requesting user intent classification, alongside the most recent message.

A worked example is provided:

A first approach is to use a separate LLM call to get a summary of the chat history including the latest message, and then provide this summary in an intent classification prompt. This technique

improves classification vs solely using a most recent message exchange between user and assistant, and also improves classification relative to using entire chat history.

Example prompt:

```
SUMMARISATION_SYSTEM_PROMPT:

   "You are an expert at summarising the current user goal based on
chat dialogue history.

  Your task is to state what the user's current goal is in relation to
the provided chat dialogue history.

  Your response should be a single sentence summarising the user's
current goal.

SUMMARISATION_USER_PROMPT:

   "Here is the user's chat dialogue history:
  {history_string}
  Now, state what the user's current goal is.
```

Alternatively, in order to make identification of 'early exit' messages easier for the LLM, the summarisation prompt may be configured to highlight the latest input:

```
"The provided chat dialogue history is in chronological order.

  Pay attention to the latest user dialogue.  "
```

Chain-of-thought prompting may be incorporated as follows:

```
"Think out loud and walk me through your thinking performing the
following steps:

     1. Analyse the provided chat dialogue history, which is in
chronological order.
```

```
2. Pay attention to the latest user dialogue.

3. Analyse what the user's current goal is.

4. Based on your conclusion from step 3, generate a single sentence
that summarises the current user goal. "
```

Providing the full chat history, either alone or in addition to the summary, may also improve performance on detecting early exit.

In summary, the LLM prompt requesting that the first LLM return a selection of a processing function may include in the LLM prompt a chat history comprising messages submitted to and received from the software service by the user (in the same session) via the messaging interface, and further requests that the first LLM summarise the chat history and use the summarized chat history in processing the request for the selection of the processing function.

Alternatively, the LLM prompt requesting that the first LLM return a selection of a processing function is preceded by a chat summary LLM prompt detailing a chat history comprising messages submitted to and received from the software service by the user via the messaging interface, and requesting that the LLM to which the chat summary prompt is input summarises the chat history; and the LLM prompt requesting that the first LLM return a selection of a processing function includes the summarized chat history and instructs the first LLM to use the summarized chat history in processing the request for the selection of the processing function. The LLM prompt to which the chat summary LLM prompt is transmitted may be the first LLM or may be a different LLM, for example, Claude.

Nuanced Classification for Unsupported Operations

Processes limit user intent classification to intents aligning with processing functions provided by the software service. There may be instances in which a user intent is classified as a closest processing function to the user intent, but that may be unintended. For example, a request to update an invoice in a way that isn't supported (deletion of line items) may cause classification

as "update_invoice" instead of “no_such_tool”, leading to parameter extraction for something that is supported (like adding line items) and unintended action.

Optionally, the articulations of the processing functions may be configured to specify one or more unsupported operations that may be semantically similar. For example, a usage articulation of the “update_invoice” processing function may state:

"Can be indicated by a user specifying things to only perform adding line items or changing status to a customer's existing invoice. It cannot perform change, modification, or deletion of line items and should not be used for that.”

In summary, the articulations of the processing functions may be configured to specify a user intent that is not aligned with the processing function. Advantageously, potential for mis-classification is suppressed.

Additionally or alternatively, the intent classification prompt (such as at S104, 504, S803) may include closely relevant examples of correct intent classification (‘few-shot’ prompting). These may include examples of supported user intents being classified to their associated processing function, and/or unsupported user intents being classified to “no_such_tool”.

These adjustments provide some improvement in classifying unsupported operations as “no_such_tool”, instead of a processing function that is semantically similar.

In summary, the LLM prompt requesting user intent may be configured to include examples relating to correct classification of both supported and unsupported operations. A user intent aligning with an unsupported operation may be handled by outputting a message to the user notifying that their intended operation is not supported.

Chain-of-thought prompting for parameter extraction

In Figure 5A a data (wherein the data may be one or more parameters which may be referred to as values, data field values, or parameter values) extraction step 510 is performed by the model 505 by creating an LLM prompt to the LLM comprising the initially received message 502 and a list of data fields to instantiate. The list is generic for the selected processing function, and may

be stored in one or both of the data graph representation of the selected processing function and the text descriptor of the selected processing function. The LLM prompt requests that the LLM extract from the message 502 values of any of the listed data fields for which values are provided in the message 502. In the worked example, the values are name="John", which is on a traversal path corresponding to a 'search invoice by name' component action, and amount="50", which is on a traversal path corresponding to an 'add late fee flat fee' component action.

In Figure 5B a data extraction step 538 is performed by the model 505. In Figure 5C a data extraction step 564 is performed by the model 505. At step 582 of Figure 5D values are extracted late fee amount and invoice number.

In the above examples and in general in parameter extraction tasks in processes, a prompt issued to the LLM to execute the parameter extraction may request the LLM to extract the parameter from a most recent message only, or may request the LLM to extract the parameter from the message history (i.e. a chat history, being a series of messages exchanged between the end user and the service 503 via the chat interface 501) corresponding to the present session or since initiation of the present processing function. That is, the LLM may be requested to extract the parameter from a series of messages.

As the chat history gets longer, simply as a consequence of there being more text, the number of elements (i.e. words, phrases, numbers, or combinations thereof) that may meet the specification for a parameter extraction, or may otherwise be extracted as a parameter, increases. Inherently therefore, a risk of extracting incorrect information in the form of incorrect parameters increases.

That risk may be reduced by the processing of the input messages from the user including a chain-of-thought prompt. An example of how such a chain-of-thought prompt may be utilised is now presented.

In a first exemplary use of chain-of-thought prompting, an emphasis is placed on a reply to a question from the service 503. An initial prompt may be composed as follows (noting that the service 503 refers to itself as 'the assistant', the plurality of data fields defined in the text descriptor of the processing function is referred to as the PARAMETER LIST, and the chat history is referred to as the CHAT DIALOGUE):

<PROMPT BEGINS>

"Function: {function_id}
Function description: {function_description}
[PARAMETER LIST START]
{parameter_list}
[PARAMETER LIST END]
[CHAT DIALOGUE START]
{chat_history}
[CHAT DIALOGUE END]
[TASK]
Interpret what the user wants to do and select the best parameters needed to execute the function. Only extract parameters mentioned in the parameter list. Think out loud and walk me through your thinking performing the following steps:

1. Identify what the user wants to do next. Pay special attention if the user is replying to a question made by the assistant on the previous step.
2. Based on the response from 1, extract the information needed from the CHAT DIALOGUE.
3. Format the parameters extracted to meet the requirements in PARAMETER LIST. Line items require three fields 'description', 'quantity', and 'amount'. You may summarise 'description' based on the CHAT DIALOGUE."

Optionally, this type of prompt is supplemented by the addition of a supplementary LLM call to introspect on the first pass parameter extraction. For example, the supplementary LLM call may be composed as follows:

"Function: {function_id}
Function description: {function_description}
[PARAMETER LIST START]
{parameter_list}
[PARAMETER LIST END]
[CHAT DIALOGUE START]
{chat_history}

[CHAT DIALOGUE END]
[PROPOSED SOLUTION START]
{chain_of_thought_response}
[PROPOSED SOLUTION END]
[TASK]
Parameters can only be extracted from the CHAT DIALOGUE. Do you think the proposed solution is correct? Your response should strictly follow the specifications in PARAMETER LIST. If a parameter is not found, do not include it in the response
Return the correct parameters as a JSON object where the keys are the parameter names, and the values are the extracted values.
This JSON object should be your only output."

The verifier (the LLM to which the supplementary LLM call is output) may then respond with what it considers the correct response, for example in JSON format.

However, the initial prompt and the supplementary could also be combined into a single chain-of-thought prompt. The following example is also refined further with some lines about continuation from a previous interaction, to try and ignore chat history when it's irrelevant.

1. Use the dialogue context to identify what the user wants to do next. Pay special attention if the user is replying to a question made by the assistant on the previous step.
2. Based on your previous response check if the user wants to continue from a previous interaction or is starting a new one.
3. Paying close attention to your previous response, find the relevant information from the CHAT DIALOGUE. Only assume information from the chat history if this is a continuation.
4. Check if the parameters you found on step 3 relate to the user's intention on step 2. Only assume information from the chat history if this is a continuation from a previous request.
5. Based on your conclusion from step 4 generate a single JSON object where the keys are the parameter names, and the values are the correct extracted values. Format the parameters names and values to strictly follow the requirements in PARAMETER LIST. Line items require three fields 'description', 'quantity', and 'amount'. You may summarise 'description' based on the CHAT

DIALOGUE.

In summary, a procedure for extracting information from an historical chat dialogue between the user and the software service (for example via a messaging interface) may comprise submitting to the LLM a chain-of-thought prompt to the LLM comprising the historical chat dialogue, a list of one or more required data field values, and a request to extract from the historical chat dialogue one or more data field value from the list.

In a particular example, the procedure for extracting information from an historical chat dialogue between the user and the software service (for example via a messaging interface) may comprise submitting to the LLM a prompt comprising the historical chat dialogue requesting that the LLM identify a next component action to execute in the processing function, from among component actions represented by feasible next graph nodes on the traversal path, and to extract from the chat history values of one or more of the plurality of data fields defined in the text descriptor of the processing function.

Optionally, the extracting is conditional upon a positive determination by the LLM that a most recent message from the user is continuing a dialogue in the historical chat dialogue. Optionally, in a distinct subsequent prompt, the software service may input to the LLM the historical chat dialogue, the extracted values, and optionally the plurality of data fields defined in the text descriptor of the processing function, and request that the LLM verify that the extracted values are correct.

Early Exit from Tool Execution

A user may wish to interrupt their current interaction or return to a previous state within the interaction (interaction being execution of a selected processing function).

A mechanism by which to facilitate such interruption is to include an early exit processing function as a selectable processing function, which would capture the user intent based on the current chat input.

When the intent is determined by the LLM to be cancellation, interruption, or equivalent, a current execution state representing the processing function being exited or interrupted is

cleared and replaced by the early exit state, and the LLM or whichever model is composing output messages is called with an indication that it needs to inform the user that they have successfully stopped execution of the previous tool.

An alternative mechanism is to have a separate LLM running in parallel to the LLM performing intent detection and whose task is to determine if the user wants to continue or abort the current tool execution based on the entire chat history. This could be accommodated by a triage agent paradigm discussed separately below.

In summary, the computer-implemented method may comprise reviewing message content received from the user via the user interface to determine whether the user wishes to interrupt or exit a processing function currently being executed, and if so, at the software service to implement an interrupt or exit procedure and to notify the user via the user interface of a successful interrupt or exit. Further, the reviewing the message content may be by generating an early exit review LLM prompt comprising the message content received from the user via the user interface and requesting the LLM determine whether the user wishes to interrupt or exit a processing function currently being executed, and receiving the determination of the LLM.

LLM Date Extraction

Tools or processing functions being executed by the software service in accordance with inputs from the end user received via the messaging interface may require or otherwise utilize temporal information such as a date or date and time indicated by the user in a received message. Owing to the unstructured, natural, language that the user is free to use to express information via the chat interface, dates may be expressed in a manner somewhat reliant upon context or otherwise requiring clarification.

Optionally, data extraction steps (such as 510, 538) may comprise, for cases in which the list of parameter values to extract includes one or more temporal values, composing a dedicated LLM prompt specialized for temporal value extraction, and sending the dedicated LLM prompt specialised for temporal value extraction to an LLM, and receiving the response. A separate information extraction LLM prompt is composed for non-temporal value extraction, and sent to an LLM, and the response received. The LLMs to which the temporal value extraction LLM prompt and non-temporal value LLM prompts are sent may be different, for example, the temporal value

extraction prompt may be sent to an LLM with a higher accuracy for temporal value extraction than the LLM to which the non-temporal value extraction prompt is sent.

The temporal value extraction prompt may request dates and/or times in a specific format, and may include current date and/or time to assist in interpreting relative terms. The temporal value extraction prompt may also include one or more definitions of date terminology that the end user may reference and a value of date and/or time defined by the said date terminology, noting that the value may be a value range. Examples of such date terminology include financial years, calendar years, or references to a specific quarter of the calendar year, or a specific date of significance such as public or religious holiday. The date terminology may be assigned definitions of date and/or time values based on geographical settings specific to the end user or to the software service.

An exemplary temporal value extraction prompt may be composed as follows:

```
'System prompt:
The current time is: 10:00:00 on Thursday 05 September, 2024
Financial years start on 01 July and end on 30 June each year.
Calendar years start on 01 January and end on 31 December each year.
System prompt:
The current time is: 10:00:00 on Thursday 05 September, 2024

Financial years start on 01 July and end on 30 June each year.
Calendar years start on 01 January and end on 31 December each year.
You are an expert in extracting dates and date ranges from text, and
then converting them to 'YYYY-MM-DD' format so they can be used as
inputs to functions. This text may contain zero, one or many dates or
date ranges.
User prompt:
Here is the text:
<<received message text>>
Dates and date ranges may be expressed in natural language, such as
'16th of July' or 'from the 1st of January' respectively.
Your task is to extract the zero, one or many dates or date ranges from
the text, and write your answer as a list of dates in 'YYYY-MM-DD'
format.
For example, if the date is '5th of June 2024', your answer should be
['2024-06-05']. If the date range is 'everything since the 4th of July
```

2024', your answer should be ['2024-07-04', None]. Write '[]' if the text contained no dates or date ranges.'

In a further example, some context regarding the task or processing function currently being executed by the software service according to inputs received via the messaging interface may be included in the prompt and may improve accuracy:

'User: In this text, the user is instructing us to update an invoice for them. The command may contain the following date information:
- The due date of an invoice
- The invoice issued date range, which indicates a period when an invoice was issued and is used to search for invoice(s)…'

Optionally, the temporal value extraction prompt may be composed according to chain of thought prompting, which is discussed above. Optionally, a single prompt may be composed comprising two requests: one as composed above for the temporal value extraction, and one for non-temporal values.

Furthermore, in formatting the output to the user or otherwise in generating the response message to be output to the user via the chat interface, a prompt may request the LLM or other model being requested to generate the output message to detail which dates were used in executing a task, for example by including:

'Start your summary by summarising the user-supplied criteria you used to fulfil the action. Then, provide a summary of the results.'

Optionally, performing component actions of a workflow or processing function represented by graph nodes on an execution path includes a data extraction task, comprising extracting one or more parameter values or data field values from user data. The user data may be one or more from among user data stored by the software service on behalf of the user, a received message in accordance with which the data extraction task is being performed, and a message history comprising historical messages from the end user relating to the execution of the workflow or processing function. The extracting the one or more parameter values or data field values may comprise composing an LLM prompt specifically requesting the LLM extract one or more temporal values defining a date and/or time, and including in the LLM prompt for reference by the LLM a current date and/or time, and optionally also a set of one or more contextual dates or date ranges and a natural language name or equivalent reference to each member of the set.

Hardware

Figure 6 is a schematic illustration of a hardware arrangement of a computing apparatus. The methods, processes, protocols, techniques, etc, described herein may be performed by apparatus having an arrangement such as illustrated in Figure 6. Apparatus having processor hardware and memory hardware described in the present specification may include one or more devices having an arrangement such as illustrated in Figure 6. A plurality of such devices may be interconnected over a network such as a Local Area Network or the internet. A cloud service including performing one or more of the methods, processes, protocols, techniques, etc, described in the present specification may be performed by one or more devices having an arrangement such as illustrated in Figure 6. The software service may be a Software As A Service (SAAS) cloud-based service implemented by interconnected servers having arrangements such as that illustrated in Figure 6. Furthermore, at the end user side, the software service may be accessed by a computing apparatus having an arrangement such as that illustrated in Figure 6.

The computing apparatus comprises a plurality of components interconnected by a bus connection. The bus connection is an exemplary form of data and/or power connection. Direct connections between components for transfer of power and/or data may be provided in addition or as alternative to the bus connection.

The computing apparatus comprises memory hardware 991 and processing hardware 993. Further components are optional according to implementation requirements, including a network interface 995, input devices 997, and a display unit 999. The display unit 999 and the processing hardware 993 may cooperate to implement a graphical user interface. The display unit 999 may be a touchscreen display unit.

The computing apparatus may be a smartphone or tablet having a touchscreen display unit.

The computing apparatus may be a laptop or other form of personal computer having a touchscreen display unit.

The memory hardware 991 stores processing instructions for execution by the processing hardware 993. The memory hardware 991 may include volatile and/or non-volatile memory. The

memory hardware 991 may store data pending processing by the processing hardware 993 and may store data resulting from processing by the processing hardware 993.

The processing hardware 993 comprises one or a plurality of interconnected and cooperative CPUs for processing data according to processing instructions stored by the memory hardware 991.

A computing apparatus may comprise one computing device according to the hardware arrangement of Figure 6, or a plurality of such devices operating in cooperation with one another. For example, in a client:server, or cloud-based arrangement.

A network interface 995 provides an interface for transmitting and receiving data over a network. Connectivity to one or more networks is provided. For example, a local area network and/or the internet. Connectivity may be wired and/or wireless.

Input devices 997 provide a mechanism to receive inputs from a user. For example, such devices may include one or more from among a mouse, a touchpad, a keyboard, an eye-gaze system, and a touch interface of a touchscreen. Inputs may be received over a network connection. For example, in the case of server computers, a user may connect to the server over a connection to another computing apparatus and provide inputs to the server using the input devices of the another computing apparatus.

A display unit 999 provides a mechanism to display data visually to a user. The display unit 999 may display user interfaces by which certain locations of the display unit become functional as buttons or other means allowing for interaction with data via an input mechanism such as a mouse. A server may connect to a display unit 999 over a network.

Accounting System

As illustrated by Figure 7, an example of a software service is an accounting system 231. The accounting system 231 may comprise one or more computing devices and/or server devices, such as one or more servers, databases, and/or processing devices in communication over a network. The accounting system 231 may be configured to provide accounting services to users, such as entities and accounts, and to maintain accounts for a plurality of entities, such as

businesses, individuals and organisations. For example, the accounting system 231 may be used by an accounting services provider such as an accountant, and used to track payer data and invoice data generated with respect to clients of the accounting services provider, such as business entities.

According to some embodiments, the accounting system 231 may comprise a cloud based server system. The cloud based server system may comprise one or more computing apparatus such as illustrated in Figure 6. The accounting system 231 may further comprise a 992 in communication with a memory 991. The processor 992 may comprise one or more data processors for executing instructions, and may comprise one or more microprocessor based platforms, central processing units (CPUs), application specific instruction set processors (ASIPs), application specific integrated circuits (ASICs), suitable integrated circuits, or other processors capable of fetching and executing instruction code as stored in the memory. The processor 992 may include an arithmetic logic unit (ALU) for mathematical and/or logical execution of instructions, such as operations performed on the data stored in internal registers of the processor.

A communications interface 237 controls input and output of data to/from the accounting system. The interface may comprise plural individual interfaces. For example, a messaging interface such as a virtual assistant, chatbot, or AI assistant.

APIs 239 are application programming interfaces, which may be RESTful APIs. APIs control flow of data within the accounting system 231 and instruct execution of software services, microservices, and other processing instructions forming part of the accounting system 231. One or more messaging interfaces may have an underlying API 239 to control the flow of information from the messaging interface to the accounting system 231 and from the accounting system 231 to the messaging interface.

The accounting system 231 may be configured to receive and/or store data related to one or more invoices issued by an entity to a client or customer. Data relating to clients (i.e. users) may be stored by the accounting system 231 internally or at a remote database 236 accessible via a network 215. Invoice data may include a unique invoice identifier, such as an invoice number. Invoice data may also include one or more of a payment date, payment deadline, payment amount, discount amount, tax amount and unique client or invoice identifier. The unique client

identifier may include one or more of the client name, client contact information such as a telephone number, a company registration number (such as an ABN or ACN) or a number generated by the accounting system 231 to uniquely identify the client.

The accounting system 231 may also be configured to store data relating to payers associated with the business entity, such as clients and customers to whom invoices are issued. Payer data may include one or more of the payer name, payer contact information such as a telephone number, a company registration number (such as an ABN or ACN) or a payer identifier such as a payer account number.

The accounting system 231 may be configured to execute functions such as reading and writing invoice and/or payer data and communicating retrieved data to segmentation rules determination server 217. This data may be communicated between the accounting system 231 and the segmentation rule determination server 217 via network 215, wired connection (not shown) and/or may be comprised within the same computer system or server (not shown).

The accounting system 231 may communicate with one or more financial institute or banking servers, such as server 233. In some embodiments, the accounting system 231 receives, from such server(s) 233, records or documents associated with data being monitored by the accounting system 231. For example, the accounting system 231 may be arranged to receive bank feeds associated with transactions to be reconciled by the accounting system 231. The financial or banking data may be imported through a bank feed and/or a user- or accountant-created document. In some embodiments, the accounting system 231 may communicate with third-party tools of the server(s) 233 via an application protocol interface (API) to receive the banking data. Third party server(s) may include external AI models such as generative AI models including LLMs, RAG AI, to which the accounting system 231 may issue prompts and receive responses in reply.

The accounting system 231 may comprise a bookkeeping system 235 for maintaining bookkeeping account information on behalf of the user based on data input by the user, or retrieved from a bank feed or another third party source 233, such as a payment processing service, relating to financial transactions in which the user is a payer or payee.

A client device 900 may have an arrangement as illustrated in Figure 6. The memory 991 of the client device 900 may store an application 234 which provides one or more interfaces for an end user to at least one of: view, add to, edit, manipulate, process, assimilate, generate reports based on, data stored by the accounting system 231 in relation to a user account of the particular user (or any data that the user is authorised to access, such as an accountant user accessing client data). The interfaces may include a messaging interface such as a chatbot, virtual assistant, or AI assistant. A communications interface 995 controls flow of data in and out of the client device 900.

Embodiments may include a computer-implemented method for execution by a software service provider providing a software service to a plurality of end users, the method comprising: receiving, via a messaging interface, a message from an end user expressing an intent; parsing the received message into a search string; inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string; generating an LLM prompt comprising the received message, and the text descriptors of only a subset of the set of processing functions, the subset being the members of the set of processing functions of which the text descriptors were ranked in the top n results for semantic proximity to the search string by the searching mechanism, n being a positive integer equal to or greater than 2, and requesting that the LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message; causing the selected processing function to execute.

Further disclosure

From Australian provisional application 2024900528; filed on 29 February 2024; in the name of Xero Limited, a computer-implemented method is disclosed that may be performed independently of, or in combination with, the computer-implemented methods, processes, programs, disclosed elsewhere in the present disclosure:

Figure 8 illustrates a computer-implemented method. The method of Figure 8 may be implemented as an alternative or in addition to the method of Figure 1. At step S801 a message is received at the software service from an end user. Details of the method are disclosed in priority-founding Australian provisional patent application 2024900528.

The software service may be hosted by one or more servers, which servers may implement one or more virtual servers. The servers may themselves store, or may be in data communication with one or more data storage servers which store, end user data specific to each end user of the software service. Access to such data is controlled, for example by end user identity, access level, organization, etc. For example, there is no data sharing across different organisations. However, end users belonging to the same organization may have access to common data, wherein access to those data may be managed based on the particular end user and parameters such as the team to which the end user belongs within the organization, level of seniority, etc.

The software service may be an online accounting software service providing user functionality relating to accountancy and bookkeeping, including contact information storage and management, invoicing, and generation of other financial documents such as purchase orders. End user content is a term employed to refer to documentation and database entries stored and maintained by the software service and accessible to the pertinent end user. The end user content may have been generated by the software service according to inputs from the end user themselves or another end user belonging to the same organization (e.g. where the software service enables data sharing between multiple users belonging to the same organization). The software service provides data storage and processing functions to registered end users. User registration means that each different end user has a unique ID within the software service.

The software service may be entirely online so that its functionality is accessed via a web browser. Alternatively or additionally, an end user application for computers, tablets, mobile phones etc, may provide at least part of the functionality of the software service, and/or one or more user interfaces for accessing the functionality of the software service.

The message may be an email, an SMS, a whatsapp message, a voice note, or some other medium for electronic transfer of information from an end user to the software service. The message may be to a generic mailbox at the software service, with the software service identifying the end user among a population of end users based on a lookup of the sender details in the message metadata in a database of end user sender details. Alternatively each end user may have a dedicated mailbox at the software service to which to address messages. In the case of a voice note, the receive step may include a speech to text process at the software service to transform the voice note from audio data to text data. The message may be to a chatbot, AI assistant, virtual assistant, or equivalent. AI assistant, chatbot, or virtual assistant, may leverage

generative AI to phrase responses to messages input by an end user. Content for inclusion in (either directly, indirectly via a link, or indirectly via description) the response to the end user may be phrased by the chatbot, AI assistant, virtual assistant, or equivalent, using content generated by the software service in performing an user content processing function such as at S805.

The messaging interface may be setup for the sole purpose of receiving instructions from end users (for example an email inbox dedicated to receiving processing instructions from one or a plurality of users), so that any received message is assumed to contain an instruction (it being implicit that a user content processing function is intended) and so to cause execution of the method of Figure 8. In another alternative, an initial filtering step or equivalent may be performed to determine whether or not the received message includes a processing instruction or equivalent, and to proceed with the remainder of the method if the message does include the processing instruction or equivalent, and not otherwise. The filtering may be performed by intent extraction performed in a deterministic way according to a set of processing rules (in a simple example, does the message include a question or a word typically in an instruction or command such as 'please', 'would you', or one of a predefined list of verbs associated with the available user content processing functions). Or conversely, messages can be filtered out where they can be identified as expressing thanks for a previously-performed processing function. Filtering is taken to mean determining whether or not to continue with processing the received message according to steps S802 onward.

The message may be a single message from the end user to the software service with an instruction to perform a function using content from the message. The message may be a forwarded message or string of messages to or from the end user from a third party (i.e. not the end user or the software service) such as a customer, potential customer, contact, etc. The message may belong to a single session comprising a series of text inputs from the end user and responses from an AI assistant, virtual assistant or chatbot, in a UI or webpage of the software service.

The message may be pre-processed, for example to remove from a thread any message other than the most recent message or most recent exchange of messages. The pre-processing may be to format the message as a text string.

The message is received by the software service from the end user via a messaging interface. The messaging interface may be within the software service, or may be external to the messaging interface such as a generic emailing interface or messaging interface such as SMS or whatsapp. The messaging interface may comprise an email inbox at the software service, and the message may comprise an email being an individual email message or a thread of email messages. Alternatively or additionally, the messaging interface may comprise an SMS or whatsapp inbox (at the software service provider), and the message is an SMS or whatsapp message or a series of whatsapp messages or SMS messages. Such messages may comprise text or may be audio messages which are transformed to text in a preprocessing step at the software service. The messaging interface may be a virtual assistant or chatbot interface conducting a multi-message text conversation with the end user.

The message may comprise natural language content. The messaging interface may be a natural language messaging interface, and may be a single-turn or a multi-turn messaging interface.

The message may comprise values of one or more parameters or data fields that are usable by the software service in performing a data processing function on behalf of the end user. For example, the parameter values or data field values extracted from the message, or from end user content identified in the message, may constrain or otherwise guide the user content processing function.

For example, the message may identify a particular customer or customer account and request a report of number of outstanding invoices and total monetary value outstanding. The user content processing function is to “get invoices”, but the data field value extracted from the message is the identity of the particular customer or customer account. In that way, the get invoices function is constrained by the identification of the customer.

The message, whether pre-processed, transformed, or otherwise modified at the software service, is included in an LLM prompt generated by the software service at S803. The LLM prompt further comprises an articulation of each of a set of candidate user content processing functions. The articulation may be referred to as a digest, a representation, a description, a summary, or an abstract. The articulation may include one or more from among: a description of the content processing function, one or more keywords which (if identified in the message) indicate an intention to cause the content processing function to be executed, one or more data fields for

which a value is required in order for the software service to perform the content processing function. Candidate denotes candidate for selection by the LLM as the user content processing function best satisfying a direction, request, or intent expressed in the received message. The articulation may include re-phrasings of any of the elements, to serve as context for the LLM in generating a response to the prompt, and optionally also to improve search accuracy at S802.

An optional search step may be performed at S802 to extract the candidate set of end user processing functions from a superset, the complete set of user content processing functions. The articulations of each member of the complete set are embedded into vectors to compose a vector space of the articulations. The vector space is searchable at S802, by a search function, for example the received message or an excerpt thereof is the search string. The search string may itself be embedded or vectorized (that is, parsed for searching) in preparation for the searching at S802. The searching step S802 may be considered an initial filtering step, to filter out irrelevant candidates.

The searching at S802 is to return a subset of the complete set of available user content processing functions from which subset the LLM is asked to make a selection of a single subset best satisfying the user intent. Thus, methods including S802 leverage a first filtering technology, traditional vectorized searching, to reduce the number of options from the membership of the complete set to the membership of the subset identified in the search. Wherein the size of the subset is a configurable global parameter under control of the software service. To give a broad idea for the purposes of illustrating utility of the method, the complete set of user content processing functions may number of the order of 100 members, and the searching step S802 may be configured to return the 10 best matches. Rather than a fixed number of members of the subset, there may be a threshold minimum value of a similarity or other metric assessing likelihood of user intent matching the respective user content processing function.

The LLM prompt requests that the LLM identify in the received message an intended user content processing function from among the set of user content processing functions. Optionally, the LLM prompt may also request that the LLM extract from the received message values for each of one or more data fields identified in the articulation of the intended user content processing function. The LLM is a generative AI model such as Claude or Claude Instant. The one or more data fields identified in the articulation may be values upon which execution of the function at S805 is dependent. For example, the message may be ‘make me a report of the number of

outstanding invoices issued to COMPANY A and total monetary value outstanding'. The user content processing function identified in the response at S804 is "get invoices", and the data field value extracted from the message is the identity of the particular customer or customer account. The get invoices function is dependent upon, or constrained by, the identification of the customer in the received message.

The LLM prompt comprises an instruction, for example, 'based on the message content, identify which of the described/articulated content processing functions the user wishes to have performed'. The LLM prompt may also comprise an instruction such as 'extract from the message content values for the data fields listed in the pertinent description/articulation'. That is, the LLM prompt instructs the LLM to perform at least one of two functions: user intent detection/identification, and data field value extraction.

A description or articulation may be composed of two elements. A first element may comprise one or more keywords, phrases, or other indicators that assist with identifying the pertinent content processing function as the user intent. For example, the first element may be an explicit command in a message forwarded to the software service by the end user, such as 'create quote', 'create invoice', or 'update contact', or may be the equivalent in a message to the end user from a contact of the end user (and which is forwarded or otherwise relayed to the software service by the end user). A second element may comprise one or more data fields required to execute the pertinent content processing function. It is noted that, for each element, the prompt includes instructions to guide the LLM as to the usage of the element. The LLM prompt asks the LLM to extract a user intent, and to extract data field values for performing or executing the content processing function.

The LLM prompt contains all of the information that the LLM requires to perform the requested function or functions. So, the LLM identifies user intent as one of the user content processing functions from among the set articulated in the prompt itself, by finding the articulation from the set of candidate user content processing functions, and there being a one:one mapping between articulations and user content processing functions. The LLM is not pre-trained in the user content processing functions. Advantageously, one or more from among edits, deletions, and modifications, can be made to the complete set of articulations without needing to re-train any AI model. The LLM prompt further includes the received message from the end user, or a representation thereof, or an excerpt thereof.

The description or articulation of each of the content processing functions may distinguish between required and optional data field values. For example, in the case of creating a draft invoice document, if a line item price or cost is specified in the message then that is extracted by the LLM, but it is not a requirement. The create invoice API to which the JSON generated by the LLM is passed may be configured to use a default line item price where none is specified in the JSON input. Required and optional data fields may serve two purposes. Firstly, they provide context for the LLM: if there are four required data fields in a particular articulation, and the LLM identifies a value for each of the required data fields in the message, then that may help the LLM to determine that the user intended processing function is the processing function mapped to the particular articulation. Furthermore, in case the LLM prompt requests that the LLM perform data extraction, the required and optional data fields act as instructions for the data extraction.

The LLM prompt comprises a list of candidate actions (i.e. user content processing functions) and an associated description or articulation of each. The LLM prompt asks the LLM to classify a user query (i.e. the received message) in terms of one of the listed actions. The LLM prompt may also ask the LLM to identify and extract any parameters (data field values) that have been submitted by the end user as part of the user query, and optionally also to generate a response saying what action it intends to take and whether any further parameters are required.

The LLM prompt is submitted to an LLM such as Claude or Claude Instant, and at S104 a response obtained. Alternatively, another LLM such as ChatGPT, Lama2 etc, may be the recipient of the LLM prompt. The format of the response may be predefined by the LLM or may be specified in the LLM prompt. The response may be in a Json format.

The LLM performs at least one of two functions in response to the prompt: user intent identification, and data field value extraction. Data fields are pieces of information required to fulfil the user intent. Wherein, the user intent is a content processing function from among the set of candidate content processing functions described/articulated in the LLM prompt. The response from the LLM comprises an indication of the intended user content processing function. The response from the LLM may also comprise the extracted values. The data fields are properties or parameters for which a value is required in order to perform the user content processing function. The content processing functions are generic functions, such as create contact, create invoice, etc, which are tailored, personalized, or otherwise limited for the specific

information contained within the message, and which specific information may be extracted by the LLM responding to the LLM prompt.

The software service may be an SaaS architecture based software service in which content processing functions are executed via one or a series of RESTful API calls. The software service stores a mapping of articulations to API calls, so that upon identification of an articulation by the LLM and return of said identification by the LLM at S104, the software service, based on the mapping, identifies an API call to execute at S805.

At S105 the software service causes the intended user content processing function to be performed. The user content processing function may be performed in accordance with one or more data filed or parameter values extracted from the message by the LLM. The software service may itself perform the user content processing function, or may trigger the user content processing function. The software service may forward the Json payload (that is, Json content) itself to a specific content processing function via an API, or may extract data from the Json payload for submission to a content processing function via an API. Thus, the content processing function is caused to be performed. The content processing function may be to create or update a contact entry in a contacts database, or to create or update a document such as an invoice or a purchase order. In case the content processing function relates to generation or update of a document for the end user, the generated document may be transmitted to the end user for use or presented to the end user within an interface of the software service for approval. The end user may approve the generated document by logging into the software service.

The end user content processing function may comprise generating content comprising a new or updated document or a new or updated database entry, or a report or other information created at S805 by performing the identified user content processing function on end user content stored by the software service. The generated content being transmitted to the end user at S807 via the messaging interface, being presented to the end user via a graphical user interface, or being stored at a storage location accessible to the end user. Content processing function is a synonym for user content processing function. The term end user is applied to indicate that the content processing function is a processing function that generates, creates, updates, modifies, reports on, manipulates, or deletes, end user content. Wherein content includes documents and database entries. For example, the user content processing function may be a processing function that assimilates end user content (i.e. documents, data, stored by the software service)

to create a report or information, creates or updates documents or database entries created or stored by the software service in a storage area accessible to the end user and specific to the end user or their organization.

The user content processing functions may require data field values extracted from the received Message to be matched to existing data field values stored in a database at the software service. For example, to match an entity identified in the message with an entity existing in a data source stored by the software service, such as a database. The data source being data stored at the software service relating to the end user and its organization. Causing the intended user content processing function to be performed in accordance with the extracted data field values may include calling a data source of the software service in real time to verify one or more of the data field values extracted from the message.

Examples of user content processing functions include:

- Creating a new contact entry in a database of contact entries specific to the end user, the database of contact entries being a component of the software service. In which case the new contact entry may comprise one or more of the data field values extracted from the received message. For example, a unique identifier of the contact entry such as contact name may be compared with a list of contact names stored in a contacts database by the software service in a contacts database stored on behalf of, or otherwise accessible to (for example by being stored on behalf of an organization to which the end user belongs) the end user.
- Updating an existing contact entry in a database of contact entries specific to the end user, the database of contact entries being a component of the software service. The updated contact entry may comprise one or more of the data field values extracted from the received message.
- Creating a new invoice document defining one or more line items and an associated cost per line item. The new invoice document may comprise one or more of the data field values extracted from the received message. For example, a unique identifier of the line item such as a line item name or line item label, descriptor, or number, may be compared with a list of line items stored in an invoicing database by the software service in an invoicing database stored on behalf of, or otherwise accessible to (for example by being stored on behalf of an organization to which the end user belongs) the end user.

- Updating an existing invoice document. The updated invoice document may comprise one or more of the data field values extracted from the received message.
- Create a report or information based on processing, extracting information from, or assimilating, end user content stored by the software service.

End user content may comprise documents and data stored by the software service on behalf of the end user.

Once the user content processing function has been performed at S805, a result may be output to the LLM in a further prompt at S806 also comprising the received message from S801 and requesting that the LLM respond with a reply to the received message based on, including, or directing the end user to, the result. At S807, the reply generated by the LLM is output to the end user via the messaging interface.

Steps S806 and S807 are illustrated in dashed lines in Figure 1 to denote that they are optional, wherein other mechanisms for outputting results may be used instead. For example, there may be no results output, simply a text string indicating that the user content processing function has been performed. Or the user content processing functions may themselves include output steps.

The LLM in question may be a generically pre-trained off-the shelf generative AI algorithm that requires no training specific to the method of Figure 1. That is, it requires no prior knowledge at the LLM of the software service or the set of user content processing functions (because they are described/articulated in the prompt). The computer-implemented method of Figure 1 may be performed with no training phase to customize the LLM (nor any other AI algorithm) for the particular implementation scenario.

The software service provider can provide end users with access to a set of user content processing functions simply by sending or forwarding a message to the software service. The message may be unstructured text and need not expressly specify which function is to be performed. AI, in the form of the LLM, surfaces user intent and performs data extraction from the message (which is forwarded to the LLM in the prompt) without undergoing any sort of pre-training specific to the task. Therefore, no training dataset is required.

In the computer-implemented method of Figure 8, once the end user has forwarded the message at S801, the process is automatic. Thus, there is no manual intervention required.

In the computer-implemented methods of Figure 8 the messaging interface may be a natural language interface so that unstructured text may be submitted to the messaging interface by the end user. The process is triggered by receipt of a natural language message from the end user at the messaging interface. The natural language messaging interface may be a chatbot, AI assistant, or virtual assistant.

Embodiments may include or otherwise incorporate a computer-implemented method for execution by a software service provider providing a software service to a plurality of end users, the method comprising: receiving message from an end user; generating a search string based on the received message; inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string; generating and outputting an LLM prompt comprising the received message, and the text descriptors of only a subset of the set of processing functions, the subset being the top n results for semantic proximity to the search string, and requesting that the LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message content; causing the selected processing function to execute.

CLAIMS

1. A computer-implemented method for execution by a software service provider providing a software service to an end user, the method comprising:

receiving, via a user interface, message content from an end user expressing an intent;

generating a search string based on the received message content;

inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string;

generating and outputting to a first LLM, an LLM prompt comprising an expression of user intent from the received message content, and the text descriptors of only a subset of the set of processing functions, the subset being the members of the set of processing functions of which the text descriptors were ranked in the top n results for semantic proximity to the search string by the searching mechanism, n being a positive integer equal to or greater than 2, and requesting that the first LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message content;

causing the selected processing function to execute.

2. The computer-implemented method according to claim 1, wherein the user interface includes a messaging interface for exchanging messages with a virtual assistant.

3. The computer-implemented method according to claim 2, wherein the message content from the end user expressing the intent is a message or messages in a message conversation between the end user and the virtual assistant.

4. The computer-implemented method according to claim 3, wherein generating the search string based on the received message content comprises:

generating a further LLM prompt comprising the received message content and one or more additional excerpts from the message conversation, and requesting an expression of user intent be extracted from the message content and one or more additional excerpts;

outputting the generated further LLM prompt to a second LLM;

parsing the reply from the second LLM into the search string.

5. The computer-implemented method according to claim 4, wherein the further LLM prompt also requests that the second LLM augment the expression of user intent with contextual information for interpreting the expression of user intent, wherein the contextual information includes one or more from among:

- an indication of a geographical region relevant to the end user;
- an indication of a domain of knowledge relevant to the end user;
- an indication of a field of commercial activity relevant to the end user;
- identifying information relevant to the end user.

6. The computer-implemented method according to claim 4 or 5, wherein:

the first LLM and the second LLM are the same LLM; or

the first LLM and the second LLM are hosted by distinct service providers.

7. The computer-implemented method according to any of the preceding claims, wherein the searching mechanism comprises:

a vector space in which vectors representing the text descriptors of each member of the set of processing functions are expressed;

a vectorizing model configured to transform the input search string from a string format to a vector representing the input search string expressed in the vector space;

wherein the searching mechanism is configured to use vector distance between the vector representing the input search string and the vectors representing the text descriptors of each member of a set of processing functions, respectively, to rank the text descriptors of each member of a set of processing functions according to semantic proximity to the search string.

8. The computer-implemented method according to any of claims 1 to 6, wherein

the searching mechanism is a full text search of the search string against the text descriptors of each member of the set of processing functions;

wherein the searching mechanism is configured to rank the text descriptors of each member of a set of processing functions according to semantic proximity to the search string based on results of the full text search.

9. The computer-implemented method according to any of the preceding claims, wherein

the searching mechanism is hosted by the software service provider as a function of, or otherwise accessible to, the user interface; or

the searching is hosted by a third party and is accessible to the software service provider as an on-demand service.

10. The computer-implemented method according to any of the preceding claims, wherein, with respect to either one, or both, of the first LLM and the second LLM:

the LLM is hosted by the software service provider and has been at least partially trained with training data including interface messaging histories of end users of the software service using the user interface or the messaging interface; or

the LLM is a generic LLM hosted by a third party and responding to received LLM prompts as an on-demand service to the software service provider.

11. The computer-implemented method according to any of the preceding claims, further comprising:

outputting, via the user interface or messaging interface, a reply to the received message content, the reply comprising, or representing, a result of the execution of the selected processing function.

12. The computer-implemented method according to any of the preceding claims, wherein

the message content comprises a sequence of messages from the end user; wherein

the search string includes elements from a plurality of different messages in the sequence.

13. The computer-implemented method according to any of the preceding claims, further comprising, following the return of the selected processing function from the first LLM and prior to or during the execution of the selected processing function, outputting, via the user interface or messaging interface, a confirmatory query to the end user requesting confirmation that the end user intends the selected processing function to be executed;

receiving a positive response to the confirmatory query from the end user;

augmenting the text descriptor of the selected processing function with the received message content from the end user expressing the intent.

14. The computer-implemented method according to any of the preceding claims, wherein the LLM prompt further comprises, or a new LLM prompt for a third LLM, being the same as the first and second LLMs or different from the first and second LLMs, is created during the execution of the processing function and comprises:

a request that the LLM extract, from the received message content, values for each of one or more data fields identified in the text descriptor of the selected processing function, and return the extracted values in response to the LLM prompt;

a member of the set of processing functions is a processing function comprising a plurality of different sequences of component actions, wherein causing the selected processing function to execute includes selecting one from among the plurality of different sequences, and causing the processing function to execute in accordance with the selected sequence; and

the selection of one from among the plurality of different sequences is based on one or more from among:

for which of the one or more data fields identified in the text descriptor of the selected processing function the LLM was able to extract a value;

for which of the one or more data fields identified in the text descriptor of the selected processing function the LLM was not able to extract a value.

15. The computer-implemented method according to claim 14, further comprising

following the selection of the sequence and prior to or during the execution of the selected processing function in accordance with the selected sequence of component actions, outputting, via the user interface or messaging interface, a confirmatory query to the end user requesting confirmation that the end user intends the selected processing function to be executed in accordance with the selected sequence of component actions;

in response to receiving a positive response to the confirmatory query from the end user, causing the processing function to execute in accordance with the selected sequence of component actions; and

in response to receiving a negative response to the confirmatory query from the end user, making a new selection of a sequence from among the plurality of different sequences, and causing the processing function to execute in accordance with the new selected sequence of component actions.

16. The computer-implemented method according to claim 14 or 15, wherein the selected sequence of component actions includes an API call, and causing the processing function in accordance with the selected sequence includes making the API call using one or more of the data field values extracted by the LLM as a parameter.

17. The computer-implemented method according to any of claims 1 to 16, wherein:

the LLM prompt requesting that the first LLM return a selection of a processing function includes in the LLM prompt a chat history comprising messages submitted to and received from the software service by the user via the messaging interface, and further requests that the first LLM summarise the chat history and use the summarized chat history in processing the request for the selection of the processing function.

18. The computer-implemented method according to any of claims 1 to 17, wherein:

the LLM prompt requesting that the first LLM return a selection of a processing function is preceded by a chat summary LLM prompt detailing a chat history comprising messages submitted to and received from the software service by the user via the messaging interface, and requesting that the an LLM to which the chat summary prompt is transmitted summarise the chat history; and

wherein the LLM prompt requesting that the first LLM return a selection of a processing function includes the summarized chat history and instructs the first LLM to use the summarized chat history in processing the request for the selection of the processing function.

19. The computer-implemented method according to any of claims 1 to 18, wherein the LLM prompt requesting that the first LLM return a selection of a processing function is configured to include examples of unsupported operations.

20. The computer-implemented method according to any of the preceding claims, wherein the causing the selected processing function to execute comprises determining and following an execution traversal path in a data graph;

wherein the data graph is a directed data graph representing the processing function, each of the component actions being represented by a graph node, and each of the plurality of different executable sequences of component actions being represented by a traversal path connecting graph nodes representing component actions in the executable sequence between a traversal path end node and a traversal path start node; and

wherein causing the selected processing function to execute comprises performing the component actions represented by graph nodes on the execution traversal path between the traversal path end node and the traversal path start node in forward or reverse sequence, starting at either the traversal path start node or the traversal path end node, and determining the execution traversal path during the execution by, when a plurality of valid next nodes in the

execution traversal path are available, selecting which of the next nodes to include in the execution traversal path in dependence upon an outcome of a component action already performed in the execution traversal path or in dependence upon information extracted from the received message content.

21. The computer-implemented method according to claim 20, wherein

each of the set of processing functions maps to a different graph traversal path end node, and

the execution traversal path is at least partially determined by identifying the graph traversal path end node mapped to by the selected processing function, and finding a starting node from which there are valid graph traversal paths terminating at the identified graph traversal path end node, and determining the execution traversal path between the starting node and the identified graph traversal path end node along one of the valid graph traversal paths.

22. The computer-implemented method according to claim 20 or 21, wherein causing the selected processing function to execute further comprises determining the execution traversal path during the execution by, when the traversal path is at a graph node from which two or more directed edges originate or at which two or more directed edges terminate, selecting which directed edge to follow in dependence upon a data field value extracted from the received message.

23. The computer-implemented method according to claim 22, wherein selecting which directed edge to follow in dependence upon a data field value extracted from the received message is dependent upon the data field instantiated by the data field value, wherein the data field is one of a plurality of data fields defined in a text descriptor of the processing function, each of the plurality of data fields corresponding to a different directed edge originating from, or terminating at, a single graph node in the data graph.

24. The computer-implemented method according to any of claims 20-23, wherein the user interface is a messaging interface of a virtual assistant, and wherein a component action represented by a graph node on the traversal path is a data retrieval component action, and wherein plural directed edges originate or terminate at the graph node representing the data retrieval component action, one of the plural directed edges being selectable in determining the execution traversal path when a number of retrieved data artefacts exceeds a predefined

threshold maximum number and terminating at a graph node representing a query user component action, the query user component action comprising outputting a query to a user via the virtual assistant to select a subset of retrieved data artefacts corresponding in size to the predefined threshold maximum number, and receiving an indication of a selected subset from the user in response to the query.

25. The computer-implemented method according to claim 24, wherein

determining the execution traversal path comprises following one or more directed edges to return to the graph node representing the data retrieval component action, and selecting which of plural directed edges originating or terminating at the graph node representing the data retrieval component action to follow in dependence upon the selected subset indicated by the user in response to the output query.

26. The computer-implemented method according to any of claims 20 to 25, wherein the user interface is a messaging interface of a virtual assistant, the virtual assistant being configured, following the selecting a processing function to execute in response to the received message, to output a confirmatory message to the end user via the messaging interface identifying or describing the selected processing function, and seeking a positive or negative reply, the virtual assistant being configured to respond to a positive reply by proceeding with causing the selected processing function to execute, and being configured to respond to a negative reply by obtaining a new expression of user intent and repeating the selecting with the new expression of user intent.

27. The computer-implemented method according to any of claims 20-26, wherein performing the component actions represented by graph nodes on the execution path in sequence includes performing one or more component actions comprising a data extraction task, the data extraction task comprising extracting information from the message content or from data stored by the software service and accessible to the end user or relating to the end user, the computer implemented method further comprising outputting to the end user via the user interface the extracted information and seeking a positive reply indicating approval of the extracted information, or a negative reply indicating disapproval of the extracted information, and in the event of receiving the positive reply, proceeding along the execution traversal path, and in the event of receiving the negative reply, either repeating the data extraction task with

different parameters or runtime variables, or obtaining replacement information from a message input to the user interface by the end user.

28. The computer-implemented method according to any of claims 20 to 27, wherein the user interface is a virtual assistant, or is a component of a virtual assistant, and wherein a component action represented by a graph node on the traversal path is a data retrieval component action, and wherein plural directed edges originate or terminate at the graph node representing the data retrieval component action, one of the plural directed edges being selectable in determining the execution traversal path when a number of retrieved data artefacts does not meet a predefined threshold minimum number and terminating at a graph node representing a query user component action, the query user component action comprising outputting a query to a user via the virtual assistant to initiate creation of one or more data artefacts to meet the predefined threshold minimum number, or to change a parameter in the data retrieval action, and a reply from the user in response to the query.

29. The computer-implemented method according to claim 28, wherein

determining the execution traversal path comprises following one or more directed edges to return to the graph node representing the data retrieval component action, and selecting which of plural directed edges originating or terminating at the graph node representing the data retrieval component action to follow in dependence upon reply from the user in response to the output query.

30. The computer-implemented method according to any of claims 20 to 29, wherein causing the selected processing function to execute further comprises determining the execution traversal path during the execution by, when the traversal path is at a graph node from which two or more directed edges originate or at which two or more directed edges terminate, selecting which of the two or more directed edges to follow in dependence upon a user intent extracted from the received message.

31. The computer-implemented method according to any of claims 20 to 30, wherein

the starting graph node is a predefined starting graph node for the processing function, or

the starting graph node is selected in dependence upon a user intent extracted from the received message.

32. The computer-implemented method according to claim 30 or 31, wherein

the user intent is extracted from the received message by submitting the received message to an LLM in a prompt requesting that the LLM select, from a plurality of processing functions described in the prompt, a processing function best matching the user intent.

33. The computer-implemented method according to any of the preceding claims, further comprising reviewing message content received from the user via the user interface to determine whether the user wishes to interrupt or exit a processing function currently being executed, and if the user does wish to interrupt or exit a processing function currently being executed, at the software service implementing an interrupt or exit procedure and notifying the user via the user interface of a successful interrupt or exit.

34. The computer-implemented method according to claim 33, wherein the reviewing the message content is by generating an early exit review LLM prompt comprising the message content received from the user via the user interface and requesting the LLM determine whether the user wishes to interrupt or exit a processing function currently being executed, and receiving the determination of the LLM.

35. The computer-implemented method according to any of the preceding claims, wherein

performing the component actions represented by graph nodes on the execution path in sequence includes performing one or more component actions comprising a data extraction task, the data extraction task comprising extracting information from the message content, and wherein executing the data extraction task includes extracting information from an historical chat dialogue between the user and the software service by inputting to the first LLM or another LLM a chain-of-thought prompt comprising the historical chat dialogue, a list of one or more required data field values, and a request to extract from the historical chat dialogue one or more data field value from the list.

36. The computer-implemented method according to any of the preceding claims, wherein

performing the component actions represented by graph nodes on the execution path in sequence includes performing one or more component actions comprising a data extraction task, the data extraction task comprising extracting information from the message content, and wherein executing the data extraction task includes inputting to the first LLM or to another LLM a

prompt comprising the historical chat dialogue requesting that the LLM identify a next component action to execute in the processing function, from among component actions represented by feasible next graph nodes on the traversal path, and to extract from the chat history values of one or more of the plurality of data fields defined in the text descriptor of the processing function.

37. The computer-implemented method according to any of the preceding claims, the method including extracting one or more parameter values or data field values from user data by composing a temporal value extraction LLM prompt specifically requesting the first LLM or another LLM extract from the user data one or more temporal values defining a date and/or time, and including in the temporal value extraction LLM prompt for reference by the LLM a current date and/or time, the user data comprising at least one of a received message, a message history between the user and the software service via the user interface, and data stored by the software service specific to the user.

38. The computer-implemented method according to claim 37, wherein the temporal value extraction LLM prompt further comprises a set of one or more contextual dates or date ranges and a natural language name or equivalent reference to each member of the set.

39. An apparatus including a computer system comprising processor hardware and memory hardware, the computer system operable to provide a software service to an end user, the memory hardware storing processing instructions which, when executed by the processor hardware, cause the computer system to perform a method including:

receiving, via a user interface, message content from an end user expressing an intent;

generating a search string based on the received message content;

inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string;

generating and outputting to a first LLM, an LLM prompt comprising an expression of user intent from the received message content, and the text descriptors of only a subset of the set of processing functions, the subset being the members of the set of processing functions of which the text descriptors were ranked in the top n results for semantic proximity to the search string by the searching mechanism, n being a positive integer equal to or greater than 2, and requesting

that the first LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message content;

causing the selected processing function to execute.

40. An apparatus including a computer system comprising processor hardware and memory hardware, the computer system operable to provide a software service to a plurality of end users, the memory hardware storing processing instructions which, when executed by the processor hardware, cause the computer system to perform a computer-implemented method according to any of claims 1 to 38.

41. A computer program comprising processing instructions which, when executed by processor hardware in a computing device, cause the computing device to perform a method including:

receiving, via a user interface, message content from an end user expressing an intent;

generating a search string based on the received message content;

inputting the search string to a searching mechanism configured to rank a text descriptor of each member of a set of processing functions according to semantic proximity to the search string;

generating and outputting to a first LLM, an LLM prompt comprising an expression of user intent from the received message content, and the text descriptors of only a subset of the set of processing functions, the subset being the members of the set of processing functions of which the text descriptors were ranked in the top n results for semantic proximity to the search string by the searching mechanism, n being a positive integer equal to or greater than 2, and requesting that the first LLM return a selection of a processing function of the subset for which the text descriptor best matches the intent expressed in the received message content;

causing the selected processing function to execute.

42. A computer program comprising processing instructions which, when executed by processor hardware in a computing device, cause the computing device to perform a computer-implemented method according to any of claims 1 to 38.

43. A non-transient computer-readable medium storing the processing instructions of claim 41 or 42.

44. A computer-implemented method executed by a software service provider providing a software service to a plurality of end users, the method comprising:

receiving, via a messaging interface, a message from an end user;

generating an LLM prompt comprising the received message, an articulation of each of a set of candidate user content processing functions, and a request that the LLM identify in the received message an intended user content processing function from among the set of user content processing functions;

receiving, from the LLM, an indication of the identified intended user content processing function;

causing the identified intended user content processing function to be performed.

45. The computer-implemented method according to claim 44, wherein:

the LLM prompt further comprises a request that the LLM extract from the received message values for each of one or more data fields identified in the articulation of the intended user content processing function;

the method further comprises:

receiving, from the LLM, the extracted values;

wherein:

the identified intended user content processing function is caused to be performed in accordance with the extracted values.

46. The computer-implemented method according to claim 44 or 45, further comprising:

obtaining a result of the performing of the identified intended user content processing function;

generating a further LLM prompt comprising the obtained result and the received message, and a request that the LLM reply with a response to the received message incorporating the result and for output to the end user via the messaging interface.

47. The computer-implemented invention according to any of claims 44 to 46, further comprising:

parsing the received message for searching;

searching a vector space composed of an articulation of each of a complete set of user content processing functions to extract from the complete set of user content processing functions a subset that are most similar to the parsed received message;

and

generating the LLM prompt with the extracted subset from the searching as the set of candidate user content processing functions.

48. The computer-implemented method according to claim 47, further comprising:

a pre-processing step comprising obtaining documentation describing each of the complete set of user content processing functions, and automatically extracting from the documentation an articulation of each of the complete set of user content processing functions, including one or more from among:

a text string describing a purpose of the user content processing function;

one or more re-phrasings of the text string describing the purpose of the user content processing functions;

one or more API calls to be executed in performing the user content processing function;

values for each of one or more data fields to be obtained from the end user via the messaging interface and according to which the user content processing function is executed.

49. The computer-implemented method according to any of claims 44 to 48, wherein

the user content processing function comprises a defined flow of atomic processing tasks, one or more of the defined flow of atomic processing tasks being dependent upon a value of a data field obtainable from the end user via the messaging interface.

50. The computer-implemented method according to any of claims 44 to 49, wherein

the messaging interface is a virtual assistant or chatbot interface conducting a multi-message conversation with the end user.

51. The computer-implemented method according to any of claims 44 to 50, wherein:

the messaging interface comprises an email inbox at the software service, and the message may comprise an email being an individual email message or a thread of email messages; or wherein

the messaging interface comprises an SMS or whatsapp inbox, and the message is an SMS or whatsapp message or a series of whatsapp messages or SMS messages; or wherein

the messaging interface comprises a speech-to-text interface for receiving from the end user, and the message is an SMS or whatsapp message or a series of whatsapp messages or SMS messages.

52. The computer-implemented method according to any of claims 45 to 51, wherein causing the intended user content processing function to be performed in accordance with the extracted values includes calling a data source of the software service in real time to verify one or more of the data field values extracted from the message.

53. The computer-implemented method according to any of claims 44 to 52, wherein the complete set of user content processing functions comprises one or more from among:

generating content comprising a new or updated document or a new or update database entry;

generating report content or information by processing and/or filtering end user content stored by the software service in relation to a user account of the end user in accordance with processing instructions specified in the user content processing function and one or more parameters or data field values extracted from a message received from the end user;

create a new contact entry in a database of contact entries specific to the end user, the database of contact entries being a component of the software service, the new contact entry comprising one or more of the data field values extracted from the received message;

update an existing contact entry in a database of contact entries specific to the end user, the database of contact entries being a component of the software service, the updated contact entry comprising one or more of the data field values extracted from the received message;

create a new invoice document defining one or more line items and an associated cost per line item, the new invoice document comprising one or more of the data field values extracted from the received message;

update an existing invoice document, the updated invoice document comprising one or more of the data field values extracted from the received message.

54. The computer-implemented method according to any of claims 44 to 53, wherein the messaging interface is a single-turn interface for causing the intended user content processing function to be performed, the single-turn comprising transmitting the message to the software service provider via the messaging interface.

55. The computer-implemented method according to any of claims 44 to 54, wherein the messaging interface is a multi-turn interface for causing the intended user content processing function to be performed, wherein the multi-turns include an initial turn of transmitting the message to the software service provider via the messaging interface, and one or more subsequent turns comprising, for each of one or more data fields identified in the articulation of the intended user content processing function:

by the software service:

outputting to the end user, via the messaging interface or via another user interface, a request for a value of a data field identified in the articulation of the intended user content processing function and for which the LLM failed to extract a value from the received message; or

outputting to the end user, via the messaging interface or via another user interface, a request to verify the value of a data field identified in the articulation of the intended user content processing function and for which the LLM extracted a value from the received message;

receiving a value of the data field from the end user, the value being a new value or a verification of value previously extracted by the LLM;

proceeding with the user content processing function in accordance with the received value or values of the data fields and one or more data field values extracted from the received message by the LLM.

56. The computer-implemented method of any of claims 1 to 38, further comprising the computer-implemented method of any of claims 44 to 55.

57. Apparatus comprising memory hardware and processor hardware, the memory hardware storing processing instructions which, when executed by the processor hardware, cause the apparatus to:

receive, via a messaging interface, a message from an end user;

generate an LLM prompt comprising the received message, an articulation of each of a set of candidate user content processing functions, and a request that the LLM identify in the received message an intended user content processing function from among the set of user content processing functions;

receive, from the LLM, an indication of the identified intended user content processing function;

cause the identified intended user content processing function to be performed.

58. A computer program comprising processing instructions which, when executed by a computing apparatus comprising processor hardware and memory hardware, cause the computing apparatus to:

receive, via a messaging interface, a message from an end user;

generate an LLM prompt comprising the received message, an articulation of each of a set of candidate user content processing functions, and a request that the LLM identify in the received message an intended user content processing function from among the set of user content processing functions;

receive, from the LLM, an indication of the identified intended user content processing function ;

cause the identified intended user content processing function to be performed.

59. A computer program comprising processing instructions which, when executed by a computing apparatus comprising processor hardware and memory hardware, cause the computing apparatus to perform the computer-implemented method of any of claims 44 to 56.