Making Generative AI Actionable

Thomas Joseph, Chief Scientist, Legion

Table of Content

What makes actionable Generative AI hard? 2

Actionable Generative AI Use Cases 3

- Query and summarization of reference documents 3
- Query and summarization of external web sites 5
- Query and summarization of application data 6
- Performing actions on the application 8
- Generating and improving communication messages 9

Summary 9

Ever since the release of ChatGPT, it seems like the entire software industry has been focused on adding generative AI to its products. And so it should be. Generative AI has the potential to be a transformational technology that changes the way we interact with computers (and one another!).

But what are people doing with generative AI? The most widely deployed forms of generative AI are large Language Models (LLMs). These excel at understanding and generating natural language. They have been used for applications where people need to read, write, or edit text, such as scanning resumes, creating job descriptions, etc.

What about taking actions at work? Finding out who is scheduled today? Changing a work assignment? Creating a new shift? At Legion, we believe that the ultimate application of generative AI will be to allow our users to interact with all aspects of our application conversationally. This is especially game-changing for our users, who spend most of their time on the move and for whom the ability to get things done just by speaking will be a huge benefit.

The reason why we don't see many examples of actionable generative AI is that it's hard. Generative AI, by definition, is designed to generate responses based on learned patterns, not necessarily to respond factually. This may be acceptable when generating a draft of a job description, for example since this will be edited by the user anyway. However, when responding to a question about scheduled hours, for example, an inaccurate answer is far less tolerable.

In this paper, we discuss some of the challenges of making generative AI actionable and some techniques we have employed to achieve this. What we describe here represents our work at a point in time. Technology continues to evolve, and our approach may be refined accordingly. We recognize that some of what we're aiming for may push generative AI beyond its current state, but at Legion, we've never shied away from taking on difficult challenges!

© 2024 Legion Technologies, Inc.   All rights reserved.

What Makes Actionable Generative AI Hard?

Simply put, generative AI is designed to generate, not regurgitate. Generative AI learns patterns and extrapolates from them. Unlike a database or search engine, it's not designed to index a set of facts and respond to them.

Another reason is that a conversational interface is often ambiguous. A graphical UI can prompt the user for unambiguous input using text boxes, drop-down lists, and so on. However, a conversational interface has to allow users to express what they want in their own words. These words vary from user to user and are often incomplete or unclear. For example, a graphical UI for a user to edit a shift can use a date picker for the user to pick the date of a shift, dropdown lists for the start time, work role, and so on. These will unambiguously specify the initial and final parameters for the shift.

On the other hand, we want to design our conversational UI to allow a user to simply say, "Extend the cashier shift on Friday by an hour." If there's only one cashier shift on Friday, nothing more should be needed. But if there's more than one shift, we want to continue the conversation by presenting options and asking the user to select one.

In addition, different users may ask for the same thing in different ways:

- "Add an hour to the cashier shift on Friday."
- "Change the end time of Friday's 10 am shift to 5 pm"
- "Make John's shift on Friday an hour longer"

Conversational UI should be able to figure out that all these are ways of asking for the same action. This is not easy.

© 2024 Legion Technologies, Inc. All rights reserved.

Actionable Generative AI Use Cases

Overcoming the challenges of actionable generative AI requires different approaches tailored to different use cases. At Legion, we are addressing the following use cases:

- Query and summarization of reference documents, e.g., asking a question and getting an answer constructed from help documents or a corporate handbook.
- Query and summarization of curated websites, e.g., asking a question and getting an answer constructed by searching an external site like the Department of Labor.
- Query and summarization of application data, e.g., asking a question about this week's schedule and getting an answer constructed from the current application data.
- Performing actions on the Legion application, e.g., instructing the application to create a new shift on a particular date and time.
- Drafting and improving the content of employee communication messages, e.g., asking for a draft of a corporate announcement.

Each of these use cases has different challenges and requires different approaches. In what follows, we discuss these challenges and the approaches we take to overcome them.

Query and summarization of reference documents

Asking a question about how to do something or about corporate policies and automatically getting a relevant and concise answer is certainly a better experience than searching through reference documents for the answer. However, if you try to use generative AI alone to do this, the results will be inconsistent and inaccurate. The reason is that generative AI is not designed to search but to generate, unlike a search engine or a database. The Large Language Model (LLM) can generate content that is not necessarily accurate (aka hallucinations).

Fortunately, we have a technique called Retrieval-Augmented Generation (RAG) that gets around this limitation. This technique combines the search capabilities of a search engine with the language generation capabilities of a LLM.

© 2024 Legion Technologies, Inc. All rights reserved.

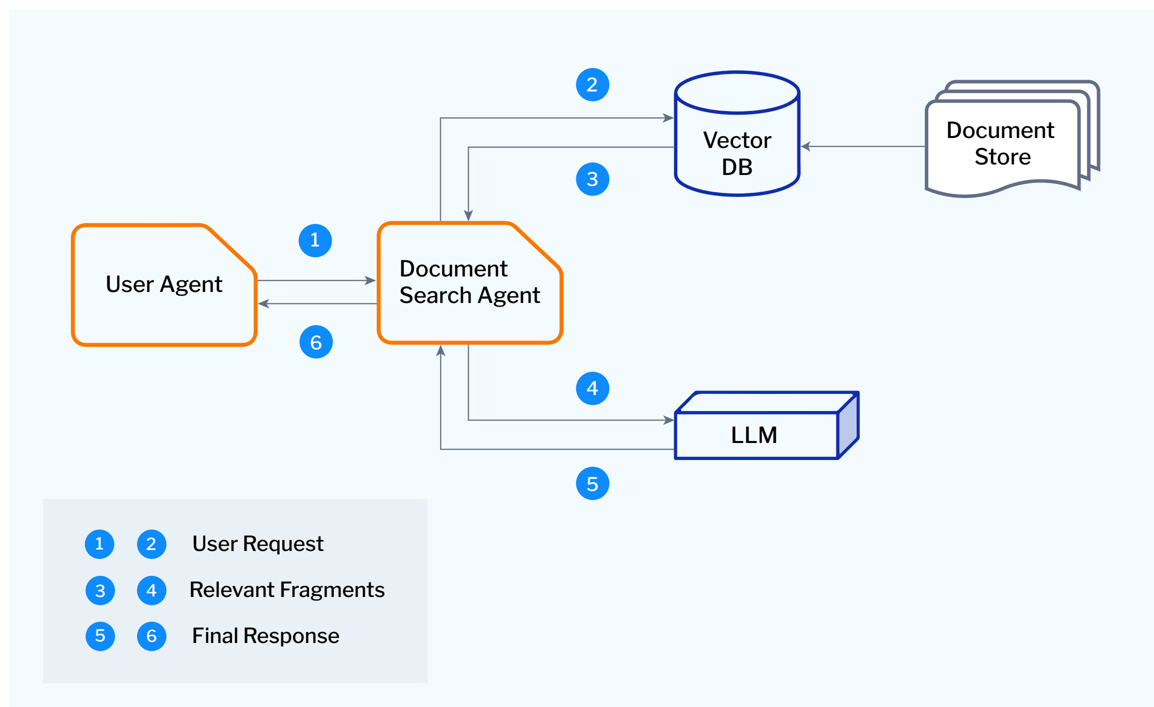
info@legion.co | +1.650.980.4873 | legion.co | @legiontech

In RAG, the documents to be searched are split into fragments and stored as vectors (aka embeddings) in a vector database. The embeddings are designed so that semantically similar fragments have vectors that are similar to one another. When a user asks a question, the question is also converted to an embedding, and the vector database is searched for similar fragments.

So, at the end of this step, we have a set of document fragments whose content matches the user's question.

The next step is to invoke an LLM with a prompt that includes the question and the retrieved fragments with instructions to generate an answer based on the retrieved fragments. Since the answer is based on the retrieved fragments vs. being generated by the LLM alone, the possibility of hallucinations is minimized.

This is shown in the figure below:



The final accuracy depends on the accuracy of the vector search in retrieving relevant document fragments and the accuracy of the LLM in generating an answer from the retrieved fragments. Using the appropriate tool for each job can keep the resulting accuracy high.

© 2024 Legion Technologies, Inc. All rights reserved.

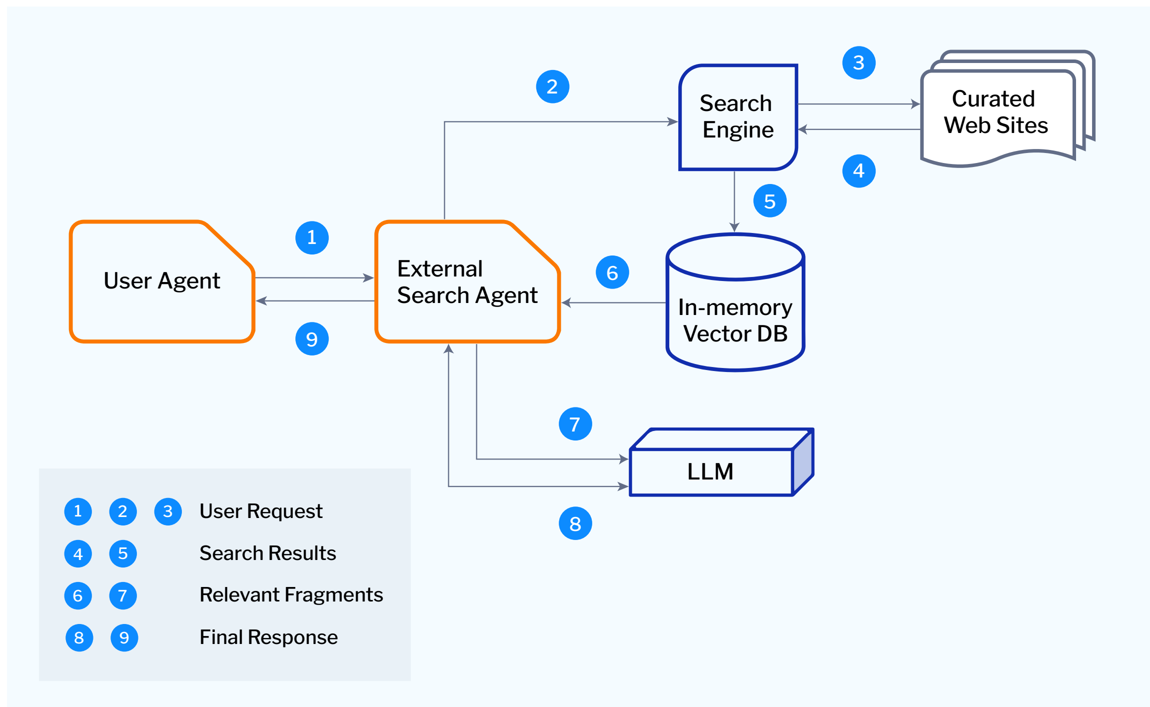
info@legion.co | +1.650.980.4873 | legion.co | @legiontech

Query and summarization of external websites

Just as it's helpful to be able to ask questions and get answers from reference documents, it's also helpful to be able to get similar answers from a set of trusted websites. In principle, this is similar to the use case above. Still, a challenge here is that the set of documents to be searched is not known upfront and so is not available as a pre-computed set of embeddings to be searched as part of the RAG process.

Our solution is to use a search engine to do a web search, then convert the documents returned into embeddings on the fly and use an in-memory vector database to search against them.

The figure below shows this:



Converting documents to embeddings on the fly has challenges, particularly if the documents are large, because this typically requires multiple invocations of an LLM and can take too long for an interactive query. We minimize this by preprocessing the retrieved documents to reduce the number of embeddings computed. We are also actively investigating approaches to improve this further.

© 2024 Legion Technologies, Inc. All rights reserved.

Query and summarization of application data

Answering questions on transactional data from an application is challenging for many reasons. First, the accuracy requirements are very high. It may be acceptable to return a response asking for clarification or even one saying that an answer is unavailable, but returning an inaccurate response is unacceptable.

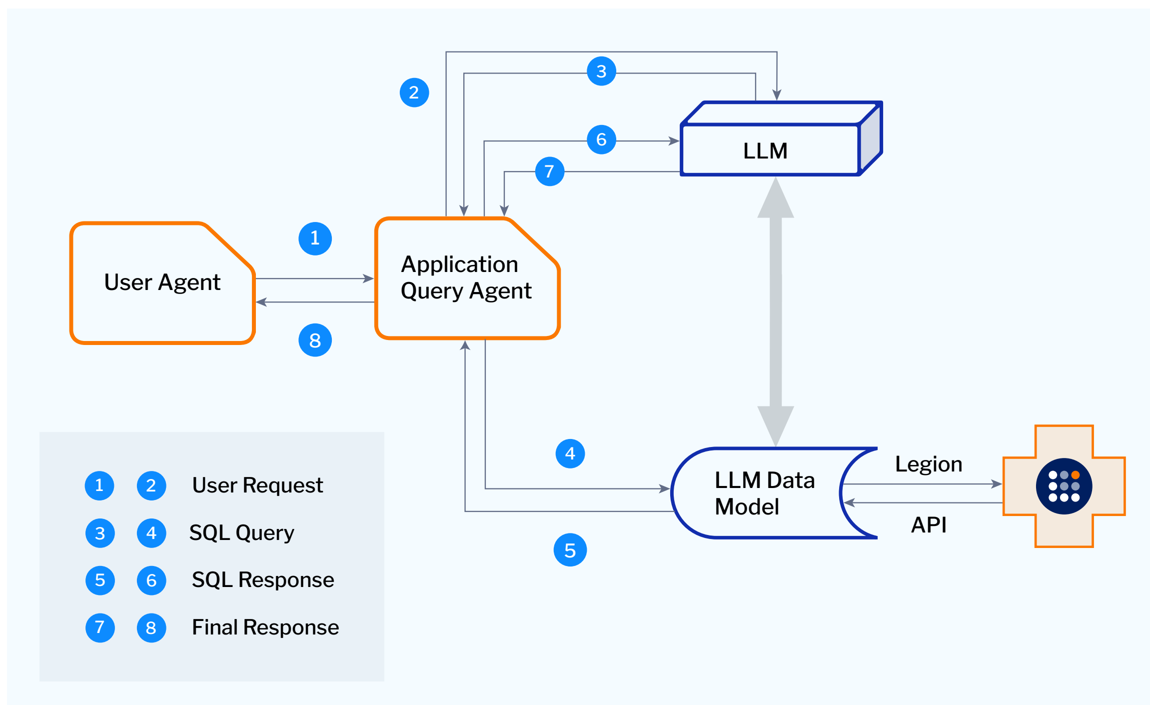
Second, users are typically restricted in what data they can access, and these restrictions must be respected when formulating an answer. An answer based even in part on data that a user should not be able to access is not acceptable.

Third, privacy restrictions limit the data sent to an external LLM and how it can be used. In most cases, sending all but minimal amounts of corporate data to an external service is unacceptable.

These challenges require a different approach for this use case. Since the LLM cannot access the application data directly, we do not ask it to query the data; instead, we ask it to tell us how we should query the data. We then run the query locally and then give just the results to the LLM to formulate a conversational answer based on the results.

This technique ensures that application data never leaves the application boundaries and remains protected by all the usual security mechanisms. What is given to the LLM instead is meta-data, i.e., a description of the structure of the data, which the LLM can use to formulate an appropriate query, e.g., in SQL. The query is then run within the application security boundary to get the results. All access to application data is done using the user's security context. In this way, the application's security guarantees are maintained.

This is shown in the figure below:



© 2024 Legion Technologies, Inc. All rights reserved.

The diagram above may look straightforward, but this technique is more complex than it seems. The data model presented to the LLM needs to be carefully designed, or the LLM will not be able to produce valid or accurate queries. While LLMs have a good understanding of SQL as a language, they do not know the semantics of the application's data model. If the data model is not designed to be easily understandable by the LLM, it will not be able to translate the user's query into valid SQL.

In general, the default application data model is not designed to easily translate natural language queries into SQL but for efficient programming. It is typically normalized and may require complex joins to obtain an answer. Attribute names are often cryptic and do not always convey the full semantics of the data. Data constraints and formats are not always obvious. While some of this information can and will be provided to an LLM as part of its prompt, there is a limit to how much information can be provided this way before the LLM begins to overlook some of it.

Another major difference is in the way entities are referenced. In a data model designed for programming, this is naturally done using unique identifiers, e.g., integers or UUIDs. In a conversational interface, users are hardly likely to refer to entities by ID. They will instead ask for entities by something more descriptive, e.g., an employee's name or the date and start time of a shift. A data model designed to be used by an LLM needs to support these ways of referencing entities.

Furthermore, the data model must be defined considering the LLM's abilities and limitations. For example, we initially found that queries based on dates fared well, but queries based on phrases like "next Friday" did not. Our LLM was not translating phrases like these accurately into the dates needed by our data model. Accuracy increased when we included a day attribute in our data model and had the LLM use this instead.

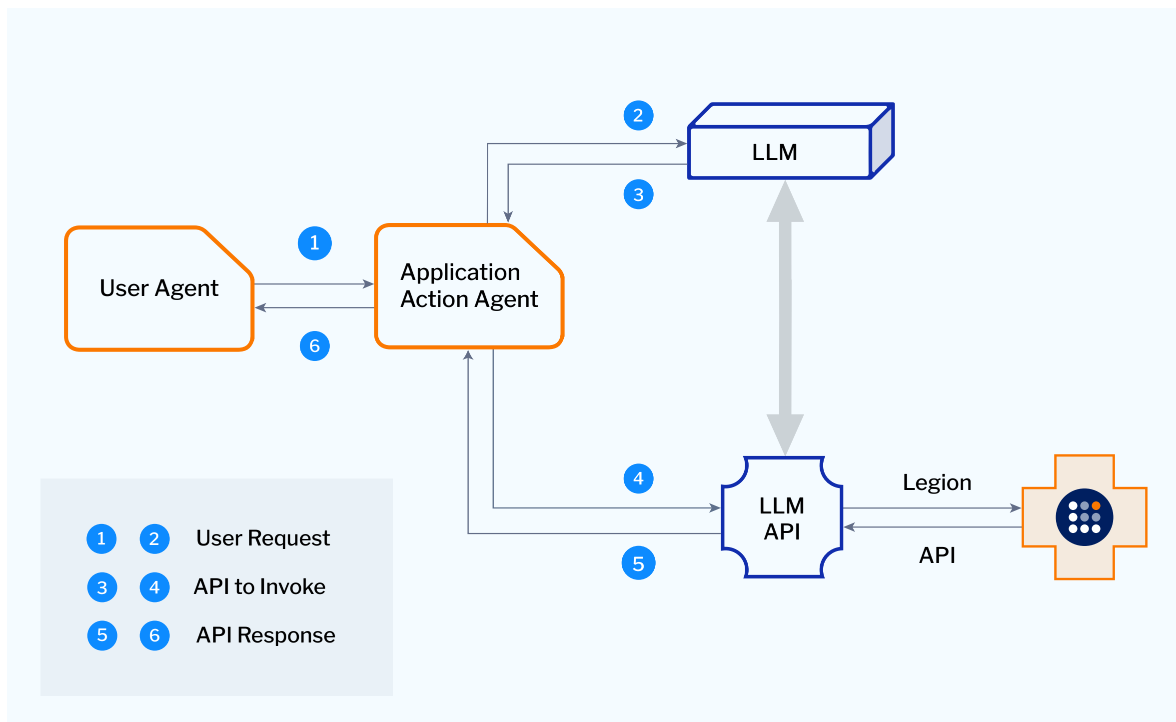
Finally, even with a well-designed LLM-friendly data model, accuracy may not reach an acceptable level. To increase accuracy further, the LLM needs to be fine-tuned with examples of likely questions and the appropriate SQL query to generate. Note that fine-tuning can and should be done with synthetic data, as what is being learned is the relationship between the user's question and the generated query, not the actual results. Actual application data should not be used for fine-tuning as this would most likely violate security guarantees.

In short, for an LLM to generate queries on application data, it needs to be presented with a data model specifically designed to translate user queries into SQL accurately and that takes into account the strengths and weaknesses of the LLM. This data model will typically differ from the default application data model and so will likely need to be built on top of the application data model. Care should be taken to ensure that data access through this model preserves the same security behavior as access to the underlying application data model. In addition, the LLM should be fine-tuned for increased accuracy.

Performing actions on the application

The challenges of performing actions on the application are similar to those of accessing application data. The LLM cannot access the application directly, so it needs to be instructed to generate a description of the action to perform from the user's request and meta-data about available actions. The actions are then performed within the application, observing all the necessary security constraints. The results of an action may optionally go through another pass through an LLM to formulate the final response.

This is shown in the figure below:



One difference from the previous use case is that the action may directly return the final response, as shown above. However, if desired, an additional step of requesting the LLM to formulate a final response can also be performed.

Once again, the API presented to the LLM for performing actions needs to be designed to make it easier to translate user requests to API invocations and to take into account the strengths and weaknesses of the LLM. The default API is unlikely to be designed with this in mind, so an LLM-friendly API must be created on top of it, preserving all security constraints.

© 2024 Legion Technologies, Inc. All rights reserved.

Generating and improving communication messages

This use case is a perfect match for the capabilities of LLMs, which are designed to generate content based on a prompt. The solution here is far simpler than for the use cases above. To generate a new message, the topic, and other attributes are sent to an LLM as part of its prompt, and the response is an initial draft. To improve on an existing message, the message is sent as part of the prompt, and the LLM responds with an improved version.

Accuracy is not a major consideration for this use case since the user will edit the message anyway. What is desired here instead is some reasonably relevant content to act as a template for the user's message.

Summary

Using generative AI for actionable use cases is challenging and requires different techniques to address the challenges of each use case. At Legion, we decided to explore the full potential of generative AI by addressing several actionable use cases described above. While there is always more that can be done, our results thus far have been promising.

About Legion

Legion Technologies delivers the industry's most innovative workforce management platform. It enables businesses to maximize labor efficiency and employee engagement simultaneously. The Legion WFM platform is intelligent, automated, and employee-centric. It's proven to deliver 13x ROI through schedule optimization, reduced attrition, increased productivity, and increased operational efficiency. Legion delivers cutting-edge technology in an easy-to-use platform and mobile app that employees love.

For more information, visit https://legion.co and follow us on LinkedIn.

© 2024 Legion Technologies, Inc. All rights reserved.
info@legion.co | +1.650.980.4873 | legion.co | @legiontech