



FastTrack for Azure

FASTTRACK FOR AZURE 15 MIN READ

Grounding LLMs



intellectronica

 MICROSOFT

Jun 09, 2023

What is Grounding?

Grounding is the process of using large language models (LLMs) with information that is use-case specific, relevant, and not available as part of the LLM's trained knowledge. It is crucial for ensuring the quality, accuracy, and relevance of the generated output. While LLMs come with a vast amount of knowledge already, this knowledge is limited and not tailored to specific use-cases. To obtain accurate and relevant output, we must provide LLMs with the necessary information. In other words, we need to "ground" the models in the context of our specific use-case.

Motivation for Grounding

The primary motivation for grounding is that LLMs are not databases, even if they possess a wealth of knowledge. They are designed to be used as general reasoning and text engines. LLMs have been trained on an extensive corpus of information, some of which has been retained, giving them a broad understanding of language, the world, reasoning, and text manipulation. However, we should use them as engines rather than stores of knowledge.

Despite their extensive knowledge, LLMs have limitations. Their knowledge is stale, as they are trained only up to a certain point in time (e.g., September 2021 for recent GPT models) and don't update continuously. Moreover, they only have access to public information and lack knowledge about anything behind corporate firewalls, private data, or use-case specific information. Consequently, we need a way to combine the general capabilities of LLMs with specific information relevant to our use-cases. Grounding provides a solution to this challenge, enabling us to leverage the power of LLMs while incorporating the necessary context and data.

Retrieval Augmented Generation (RAG) is the primary technique for grounding and the only one I will discuss in detail. RAG is a process for retrieving information relevant to a task, providing it to the language model along with a prompt, and relying on the model to use this specific information when responding. While sometimes used interchangeably with grounding, RAG is a distinct technique, albeit with some overlap. It is a powerful and easy-to-use method, applicable to many use-cases.

Fine-tuning, on the other hand, is an "honourable mention" when it comes to grounding. It involves orchestrating additional training steps to create a new version of the model that builds on the general training and infuses the model with task-relevant information. In the past, when we had less capable models, fine-tuning was more prevalent. However, it has become less relevant as time-consuming, expensive, and not offering a significant advantage in many scenarios.

The general consensus among experts in the field is that fine-tuning typically results in only a 1-2% improvement in accuracy (depending on how accuracy is defined). While there may be specific scenarios where fine-tuning offers more significant gains, it should be considered a last-resort option for optimisation, rather than the starting go-to technique. Often, customers approach us with the intention of embarking on a fine-tuning project, but we recommend they first explore the possibilities of RAG before resorting to fine-tuning.

Use-cases for Grounding / Retrieval-Augmented Generation (RAG)

Grounding, particularly with retrieval-augmented generation, has a wide range of applications. One obvious use case is search and question-answering (Q&A) systems. For instance, when you interact with Bing chat, it transparently retrieves search results and uses them to ground its responses. Similarly, many users are now building systems that leverage large language models (LLMs) to distil and make sense of information from their repositories, enabling Q&A over documents.

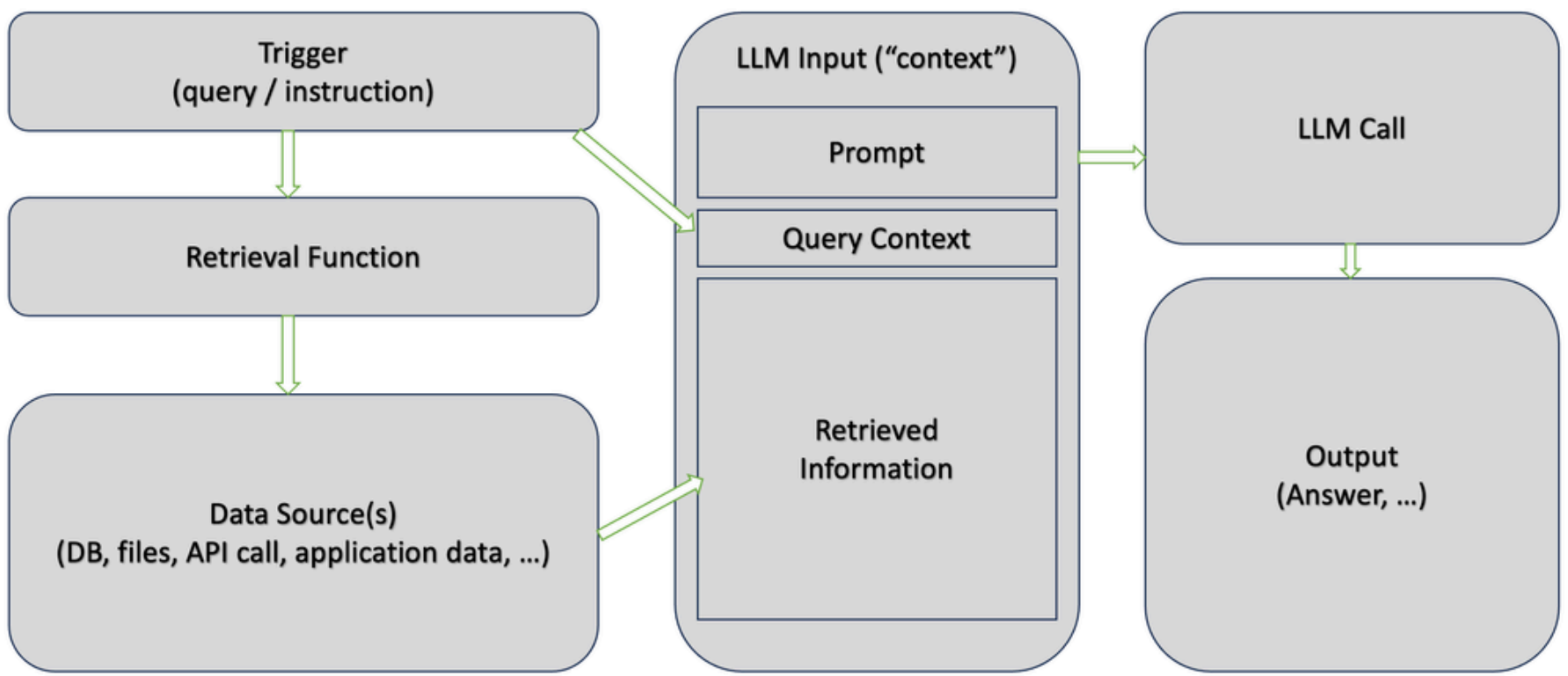
Another use case is generating content with information from an app or user interface. For example, Microsoft's Copilot, integrated with Visual Studio Code, uses an LLM to provide context-aware suggestions based on the document you are working on. This approach is also being implemented in Microsoft 365 and Power Platform, and developers are now being invited to create their own context-aware applications.

Retrieving information from APIs or external sources, such as the weather forecast or stock quotes, is another area where grounding can be beneficial. Additionally, grounding can be used for memory and state management in multi-step generation processes. Since LLMs are stateless, incorporating previous interactions or generated content can help produce more contextually relevant responses.

A Simple Retrieval-Augmented Generation Model

A basic retrieval-augmented generation model begins with a trigger, such as a user query or instruction. This trigger is sent to a retrieval function, which fetches relevant content based on the query. The retrieved content is then merged back into the context window of the LLM, along with the input prompt and the query itself. Care is taken to leave enough space for the model's response.

Finally, the LLM generates an output based on the combined input and retrieved content. This simple yet effective approach often yields impressive results, demonstrating the value of grounding in practical applications.

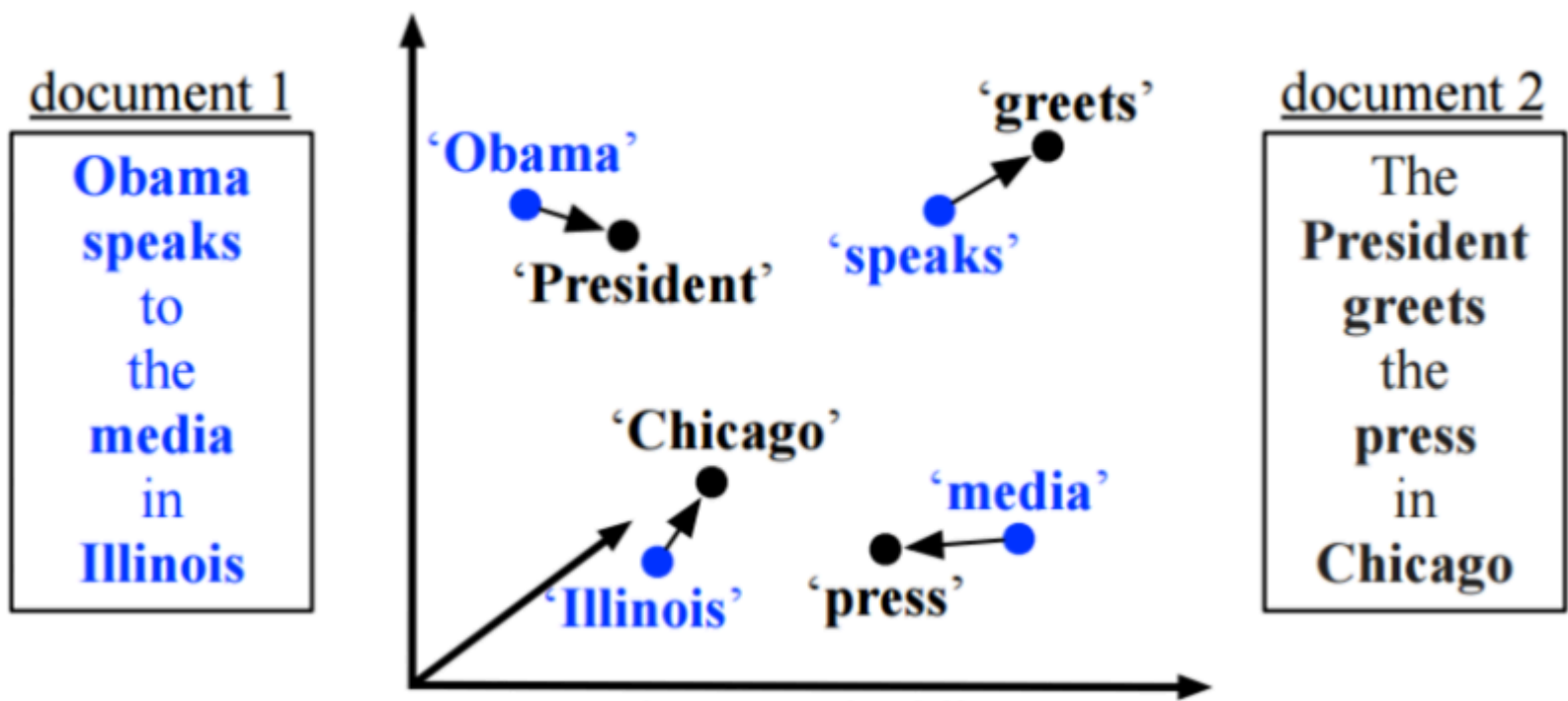


Semantic Search

Semantic search has become a primary technique for retrieval augmented generation (RAG), although it is not the only one. The process involves indexing documents or fragments of documents based on their semantic representation using embeddings. During retrieval time, a similarity search is performed from the semantic representation of the query to find the most relevant documents. This powerful, easy-to-use, and fast technique relies on an embedding model, vector index, and similarity search.

Embeddings are multi-dimensional numerical representations of "meaning" produced by language models. Given a text input, the output is a vector consisting of hundreds to thousands of numbers. A popular model for this purpose is OpenAI's text-embedding-ada-002, although there are other options, including open-source alternatives that may not be as powerful or easy-to-use.

To better understand embeddings, imagine a three-dimensional space where the location of "Obama" is close to the location of "president", and "Illinois" is close to the location of "Chicago". Comparing the documents "Obama speaks to the media in Illinois" and "The president greets the press in Chicago" would yield a high degree of semantic similarity, as opposed to comparing the first document with "The quick brown fox jumped over the whatever". The idea is to represent the semantics of text in a multi-dimensional space, allowing for efficient and accurate semantic search.



Vector Indexes and Databases

Vector indexes and databases are essential tools in the world of natural language processing. These systems store documents and index them using vector representations, or embeddings, which allows for efficient similarity searches and document retrieval. There is a wide variety of products and features available in this space, and it can be challenging to determine which one is best suited for your needs.

Dedicated vector databases such as Pinecone, Weaviate, QDrant, Milvus, and Chroma have emerged in recent years. Pinecone, for example, is a successful software-as-a-service solution, while open-source databases like Weaviate, QDrant, and Milvus can be easily deployed and often come with cloud services, and often have their own SaaS offering. Chroma, on the other hand, is the SQLite of vector databases, making it an excellent choice for simple projects with minimal scaling requirements.

Many established products have also added vector index support. Elasticsearch, for instance, is a comprehensive and feature-rich offering. Although it can be somewhat more effortful to operate, it is a natural choice for users who already rely on a large Elasticsearch cluster. Redis and Postgres, as well as many other databases, also support vector indexing, natively or via add-ons.

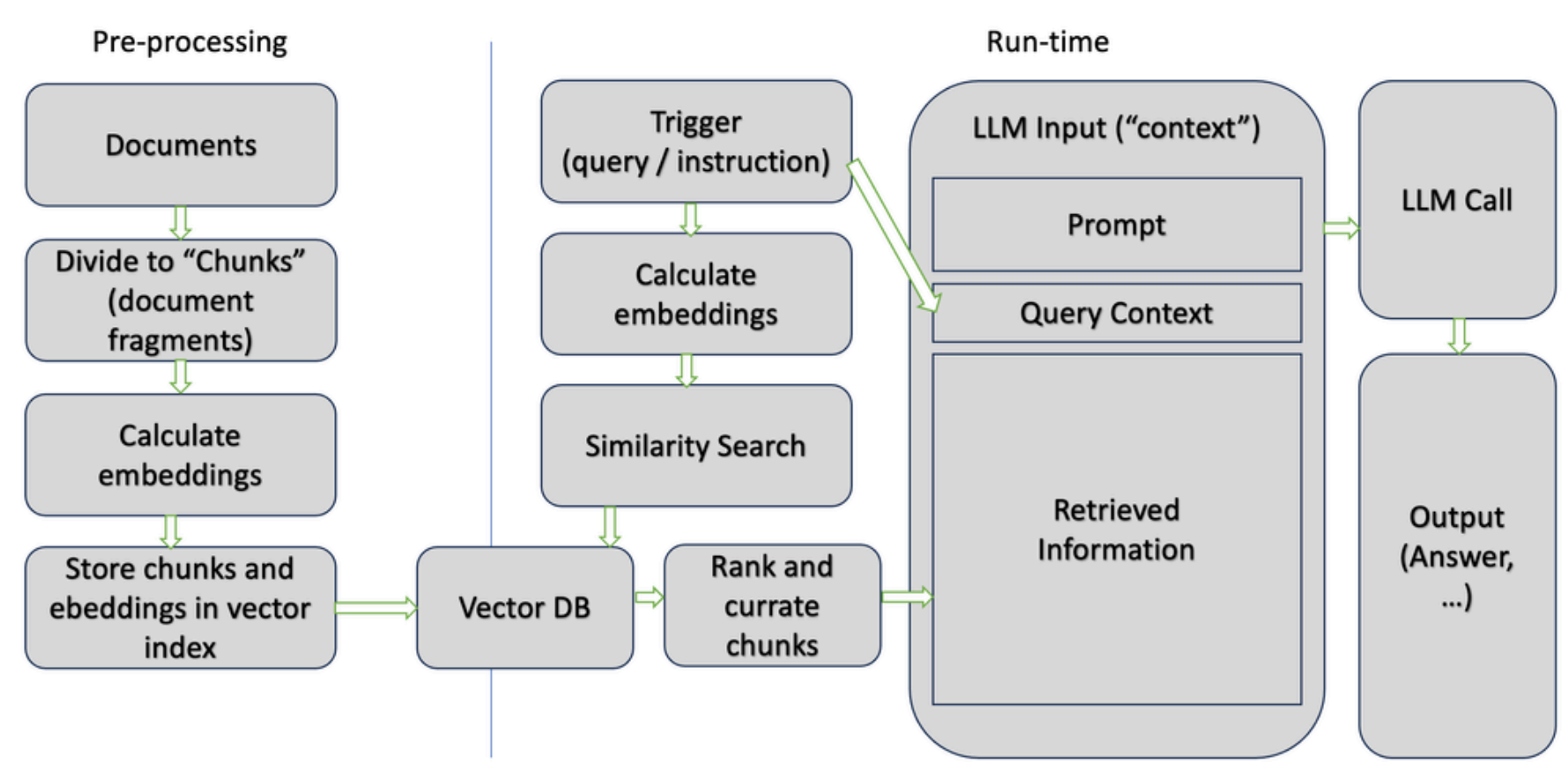
Microsoft Azure offers several preview products in this space, including Cognitive Search, Prompt Flow, and Cosmos DB. Cognitive Search, while not as straightforward to use as some dedicated vector databases, integrates seamlessly with existing Microsoft Cloud infrastructure. Prompt Flow, part of AI Studio, provides a user-friendly shortcut for data retrieval, while Cosmos DB now includes a vector index over a general document database.

When choosing a vector index or database solution, consider factors such as scaling, performance, ease of use, and your current implementation. If you have a large number of documents and users, you'll need a solution that scales well and offers robust performance. However, if your use case is less demanding, prioritize ease of use and compatibility with your preferred libraries and tools. Ultimately, the right choice will depend on your specific needs and existing infrastructure.

Simple Vector Indexing and Retrieval Model

Let's break down a simple vector-index retrieval-augmented-generation model into its preprocessing and runtime components. In the preprocessing stage, we take documents and divide them into chunks, or fragments. This is done for two reasons: to index more specific information and to fit the limited context window of the language model. For each chunk, we calculate embeddings using, for example, OpenAI's models, and store the chunks and their embeddings in a vector index.

During runtime, we receive a query or instruction and calculate embeddings based on it. We then run a similarity search using our vector database to find documents that are semantically close to our query. We rank and curate these documents, taking into account the limits of the context window, and construct the prompt to be sent to the language model.



Vector Indexing and Retrieval Techniques

There are several techniques to consider when implementing a vector indexing and retrieval model. One crucial aspect is chunking. The quality of the information retrieved from the language model can be heavily influenced by the preprocessing and chunking strategy. The optimal chunk size is a balance between small and specific chunks and larger, more comprehensive ones.

Another consideration is the format of the chunks. Decide whether to remove markup or include metadata in the text you're indexing. This may require experimentation to determine what works best for your specific use case.

Lastly, consider the retrieval scheme. One common approach is to rank documents by their similarity to the query and fill the context window with as many documents as possible. However, this may not always be the best strategy, as including less relevant documents could negatively impact the output. Instead, consider setting a similarity cutoff point to ensure that only relevant information is included.

Vector Indexing / Retrieval Tools

When it comes to orchestrating and building vector indexing and retrieval systems, there are several tools available. Semantic Kernel is Microsoft's library, available in C# .NET and Python. It contains building blocks for integrating vector index retrieval, making it a solid option for those familiar with our framework.

LangChain, on the other hand, is the industry's darling. Often referred to as the "Ruby on Rails of LLMs," this Python and JavaScript library offers a plethora of features, including various chunking models, integration of retrieval into generation, and a unified interface to vector databases. While it may seem like a one-stop-shop, it's essential to understand the underlying processes and not rely solely on the library's capabilities.

Lastly, LlamaIndex is a Python library specifically designed for vector index retrieval. While it faces stiff competition from LangChain, it remains an accessible and easy-to-use option for those looking to dip their toes into the world of vector indexing.

Vector Indexing / Retrieval Challenges

Despite the availability of powerful tools, vector indexing and retrieval can still present challenges. For instance, retrieved chunks may be semantically related but make little sense when combined. This issue is more likely to occur when dealing with broad questions rather than specific information.

Novel information, such as proper names, can also pose difficulties when creating embeddings. While language models can create embeddings for every token, the resulting embeddings may lack meaning if the model doesn't understand the token's significance.

Differentiating and ranking retrieved chunks can be another hurdle. Often, you'll encounter numerous chunks with roughly equal similarity, making it difficult to decide which ones to include. In these cases, a more sophisticated approach may be required to determine the best course of action.

Finally, it's worth noting that large amounts of information can be challenging to fit within the LLMs context window, which is limited in size. This limitation can impact the effectiveness of your vector indexing and retrieval system, so it's essential to keep it in mind when designing your solution.

Vector Indexing / Retrieval Complex Solutions

In addition to the simpler solutions previously discussed, there are more complex approaches worth considering. One such method is summarising chunks of text, either during preprocessing or at retrieval time. By shortening a paragraph-length chunk to a single sentence, the information becomes easier to work with, and more summaries can be included in the analysis.

Another technique, sometimes called "Map-reduce" (with a cheeky nod to distributed systems), involves iteratively summarising and "folding" information from chunks to create a coherent block of text suitable for feeding to the LLM. This method allows for the extraction of key information from large amounts of data.

Instead of using the original query from the user, it might be beneficial to utilise the LLM to craft a more effective query, or even multiple queries, to better answer the question at hand. This additional step can lead to improved retrieval results.

Finally, combining ranking from similarity search with other retrieval methods, such as traditional keyword search or metadata filtering, can greatly enhance the search process.

Search and Retrieval -- Working with Metadata

Metadata can be a powerful tool for refining search results. This valuable information is already known, unlike embeddings which involve an element of guesswork. Some examples of metadata that can be employed to refine search results include:

- Date: Prioritize newer documents or focus on documents from a specific time period.
- Tags / Categories: Limit or prioritize results based on relevant tags or categories, as they have already been identified and classified.
- Source / Author: Focus on results from particular sources or authors to ensure relevance.

By incorporating metadata into the search process, it becomes possible to limit the similarity search or boost rankings, leading to more accurate and relevant results.

Synthetic Metadata for Search and Retrieval

Even when metadata is not readily available, we can synthesise it during preprocessing using large language models (LLMs). For instance, LLMs can assign relevant categories or tags and generate summaries or topics for the content. This can lead to more accurate results, as LLMs, much like humans, can benefit from breaking tasks down and separating them. By doing so, different aspects of the content can be handled more effectively, ultimately improving the search experience.

Keyword Search in a Semantic World

Semantic search is powerful and often feels like magic. However, there are instances where matching specific strings is still useful. For example, when searching for a proper name or a specific phrase, a keyword search might be more appropriate than a similarity search. This approach allows users to find documents that contain the exact name or phrase they are looking for, ensuring that the results are relevant to their query.

Retrieval using APIs and Plugins

Search is not the only technique for retrieving information. We can also use internal or external APIs to obtain relevant data. By making API calls and preparing the results as a block of text, we can feed this information into the LLM's context window. Additionally, live application data can be used to drive "co-pilot" setups, further enhancing the retrieval process.

GPT plugins are an emerging standard for APIs exposed to LLMs. These plugins can be easily integrated with your application using tools such as LangChain or Semantic Kernel, or by making simple REST calls. This provides a versatile toolbox for enhancing the search and retrieval capabilities of your application.

Ranking and Set Compilation

When working with large language models (LLMs) like GPT-3.5 or GPT-4, we face a limitation in the context window size. This means that we must carefully select the information to include, as the available space is limited by the model's token budget.

One approach is to rank and limit the retrieved documents by their relevance, similarity, or other scores. However, this may result in too much or not enough information being included. To overcome this, we can pre-process or post-process the retrieved information to fit it within the context window. Techniques like creating summaries or combining different texts semantically can help build a well-suited set for generation.

Order Matters

While LLMs can cope with a certain degree of disorder (with GPT-4 being more tolerant than GPT-3.5 or simpler models), it's essential to consider the order in which information is inserted. For instance, when retrieving several chunks from a single document that only make sense in their original order, it's crucial to insert them in the correct sequence. If the text doesn't make sense to a human reader, it likely won't make sense to an LLM either.

In cases where order is important, it may be helpful to record the order in advance during indexing and use it later. This can ensure that the LLM generates coherent and meaningful output, making the most of its limited context window.

Formatting and Metadata

When working with LLMs, it's essential to remember that they are quite sensitive to formatting. This sensitivity can be advantageous, as we can use formatting to provide hints about the structure of the content. For instance, if we have several distinct pieces of text from different sources, it makes sense to separate them with newlines or other textual separators. Including metadata, such as the author or date of a document, can also be helpful for the LLM to understand the context of the information.

In cases where we need to be especially precise about structure, we can use formats like JSON or YAML. LLMs are adept at parsing and interpreting these formats, providing a clear indication of the content structure.

Tradeoffs: Speed vs Cost vs Quality/Accuracy

In any system we build or work with, there are always tradeoffs to consider. For LLMs, these tradeoffs often involve balancing speed, cost, and quality or accuracy. Quality is a somewhat elusive metric, often defined as "you know it when you see it". Speed, on the other hand, is crucial for interactive applications, as waiting for the computer to respond can be quite frustrating for users.

The cost of individual LLM calls may seem negligible at first glance, but in the aggregate, these costs can add up and significantly increase the expenses of running an application. This is especially true when compared to the costs of running more traditional, non-LLM-based applications.

Tradeoffs: Preprocessing vs Runtime

When dealing with complex setups that require multiple operations on documents, it's worth considering the tradeoffs between preprocessing and runtime. By shifting some work to preprocessing, you can potentially achieve both cost reduction and speed improvements. However, this requires a solid understanding of the content and usage patterns to make the most of this optimization technique.

It's important to note that preprocessing isn't always a cost-saving measure. For instance, if you have a million documents and only need to access a thousand of them, but you don't know in advance which ones, preprocessing all of them might not be cheaper. Nevertheless, you might still choose to invest in preprocessing to attain the desired speed improvements.

Cost

When it comes to cost, your choice of model can have a significant impact. GPT-4, for example, is very powerful and boasts a larger context window, but it comes at a higher price. On the other hand, GPT-3.5-turbo is more affordable but has a smaller context window and is less capable of handling complex tasks.

If you need to make multiple calls (either ahead of time or in real-time), remember that you don't always have to use the same model. Assess whether some of the simpler tasks can be performed effectively by a less sophisticated model, and reserve the heavyweights like GPT-4 for when they're truly necessary.

Additionally, consider whether you can complete a task with a single LLM call or if multiple calls are required. The number of calls multiplies for each task, which can lead to increased costs. Balancing these factors will help you optimize your system for both performance and cost-efficiency.

Speed

When it comes to speed, model choice is again significant. Although GPT-4 is a powerful model, it is considerably slower than GPT-3.5 Turbo. If GPT-3.5 Turbo can provide the same quality for your specific task, it might be sufficient and considerably faster.

Another factor to consider is whether you should make one call or multiple calls to the AI. Sequential GPT-4 calls can take several minutes to complete a single task. If possible, try to parallelise some of your calls. For instance, if you need to summarise multiple chunks or issue multiple queries, running them in parallel can save time. Furthermore, preprocessing and persisting results ahead of time can significantly improve speed.

Quality and Accuracy

In terms of quality and accuracy, GPT-4 is the best choice if you can afford its slowness and relatively high cost. GPT-3.5 and simpler models may work well for some tasks but struggle with others, particularly when dealing with disorder and complexity.

Evaluating the performance of these models can be challenging. To ensure a more rigorous assessment, adopt a "scientific" approach by creating a benchmark set of tasks. Utilise a combination of LLM calls and human inspection to evaluate different models and "hyper-parameters". LLMs can even be used to generate questions for evaluation purposes.

By investing in a proper evaluation framework and using tools available online or in libraries, you can gain valuable insights into the quality and accuracy of the models being used for your tasks.

Summary

Grounding, particularly with the use of RAG, is a fundamental aspect of developing valuable applications and services using large language models (LLMs). While simple vector search scenarios can be relatively straightforward to implement and yield satisfactory results, enhancing the quality and designing more intricate applications necessitates a considerable amount of thought, experimentation, and the employment of various tools and techniques.

It is essential to maintain a balance between cost, speed, and accuracy when devising a solution. Utilising multiple calls can improve the quality of the outcome, while preprocessing techniques can contribute to increased speed and cost efficiency. As you embark on your grounding journey, remember to keep these trade-offs in mind and, of course, Happy Grounding!

Addendum: using GPT-4 to produce an article from a talk

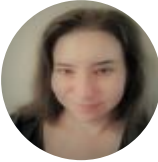
This article originates in a talk I gave recently at a Microsoft-internal event. The talk was well received, and I was encouraged to share it in article form. Armed with a recording of the talk and with my slide deck, I set to explore what's possible with GPT-4. I transcribed the talk with the OpenAI Whisper model, converted the slide deck to Markdown, edited the result to interleave the slide content with the relevant part of the transcript, and wrote a tiny script to feed those one by one to GPT-4, instructing it to rewrite it as a blog article. The result is the article above with only minimal edits!


x-posted on: everything.intellectronica.net/p/grounding-llms

Updated Jun 10, 2023 VERSION 5.0

DATA & AI

 45  Comment




intellectronica  MICROSOFT
Joined January 03, 2023
[View Profile](#)




FastTrack for Azure
Follow this blog board to get notified when there's new activity

15 Comments Newest


▼  **calaba** Copper Contributor ...
[May 14, 2025](#)
Great article.

Would it make sense to assume that when creating embeddings from relevant reterieved documents in the RAG process to conduct the similarity scoring we should use the same (=compatible) embedding model as the LLM-model we are gonna forward the query to (with the retrieved context (documents)) ??? (i.e., if using the above mentioned ada embedding model of OpenAI it should produce more relevant output using for the final answer the OpenAI ChatGPT-xx model(s) as they were a built on top of same embeddings (meanings) ??? Or is this assumption wrong / marginal ?

 0  Reply

▼  **HadiToori** Copper Contributor ...
[Jun 26, 2024](#)
Great insightful article [intellectronica](#)
How semantic index ranks documents in Copilot M365 search research within the organization?
How can we manipulate the search results of Copilot M365 for our organization, so that only the documents with certain metadata or format or tagging shows on top of the search results?
Regards


 0  Reply

▼  **lainD2** Copper Contributor ...
[Dec 15, 2023](#)
[sk3ffington](#) you're missing the point about Obama & President. It is a perfect example. Those two words were written a lot & are in the data. It might not be the answer to 'who is the current president' but the AI is going to have 'President Obama' in its ranked results. If you were to ask an AI that hadn't been grounded with who the current president was I suspect it might well think Obama given how much was written about him (2 terms worth of data created - Biden still in first). I recommend this on embeddings: [Word Embedding Demo: Tutorial \(cmu.edu\)](#)

 0  Reply

sk3ffington Copper Contributor

▼



Nov 25, 2023

...

I've been in advanced IT and cybersecurity professional for around 50 years. Fairly good article, although Microsoft proprietary acronyms should be severely limited nor used unless absolutely necessary when it comes to modern AI Models, it causes too much confusion and extra work figuring out what they mean. The IT world and now AI already have far too many acronyms to keep track of. In addition, Obama is no longer the current President, (2023) and using him as an example in the article is somewhat disturbing.

👍 0

💬 Reply

▼



nilewilson  MICROSOFT

Oct 04, 2023


...

Great breakdown, thank you!

👍 0

💬 Reply

▼



Dean Gross Bronze Contributor

Aug 14, 2023


...

I notice that neither security, compliance or privacy are mentioned. Can you please provide some guidance about how these important topics should be considered when grounding LLMs.

👍 0

💬 Reply

▼



abymmathew Copper Contributor

Jul 27, 2023

...

I am in the process of designing a custom vector database and trying to use a library that seem to be Microsoft supported. Wanted to know the author, for discussion.


The library is around hnsw -

<https://github.com/curiosity-ai/hnsw-sharp>

👍 0

💬 Reply

▼



luis woosun Copper Contributor

Jul 19, 2023

...


[intellectronica](#)


Is there a page where I can learn more about the Semantic index for copilot product? I couldn't find it. Previously, I thought that copilot only used graph API, but is the copilot searching Semantic index for copilot? So it looks like to the "Synthetic Metadata for Search and Retrieval" part mentioned above.

👍 0

💬 Reply

▼



intellectronica  MICROSOFT

Jul 19, 2023


...

[luis woosun](#) Yes, that's one of several products we offer that make it super easy and powerful to index your data and use it in RAG.

👍 0

💬 Reply

▼



luis woosun Copper Contributor

Jul 19, 2023

...

It's a great article and I read it well.

I have a question, is the semantic index for Copilot related to the Vector Indexing mentioned above?

<https://www.youtube.com/watch?v=KtsVRCsdvoU>

 0  Reply

▼ [Show More](#)

What's new

- Surface Pro 9
- Surface Laptop 5
- Surface Studio 2+
- Surface Laptop Go 2
- Surface Laptop Studio
- Surface Duo 2
- Microsoft 365
- Windows 11 apps

Microsoft Store

- Account profile
- Download Center
- Microsoft Store support
- Returns
- Order tracking
- Virtual workshops and training
- Microsoft Store Promise
- Flexible Payments

Education

- Microsoft in education
- Devices for education
- Microsoft Teams for Education
- Microsoft 365 Education
- Education consultation appointment
- Educator training and development
- Deals for students and parents
- Azure for students

Business

- Microsoft Cloud
- Microsoft Security
- Dynamics 365
- Microsoft 365
- Microsoft Power Platform
- Microsoft Teams
- Microsoft Industry
- Small Business

Developer & IT

- Azure

- Developer Center
- Documentation
- Microsoft Learn
- Microsoft Tech Community
- Azure Marketplace
- AppSource
- Visual Studio

Company

- Careers
- About Microsoft
- Company news
- Privacy at Microsoft
- Investors
- Diversity and inclusion
- Accessibility
- Sustainability



Your Privacy Choices

[Sitemap](#)

[Contact Microsoft](#)

[Privacy](#)

[Manage cookies](#)

[Terms of use](#)

[Trademarks](#)

[Safety & eco](#)

[About our ads](#)

[© Microsoft 2024](#)