Getting started with Large Language Models

[aka.ms/llm-start-here](https://aka.ms/llm-start-here)[chrisq@microsoft.com](mailto:chrisq@microsoft.com), February 2023

|  |
| --- |
| Note – this document is a work in progress. Please leave comments, ask questions, and include suggestions. |

# Background

In January 2023, everyone is buzzing about ChatGPT. This large language model can [help design programming languages](https://judehunter.dev/blog/chatgpt-helped-me-design-a-brand-new-programming-language), help you write business plans, author piano compositions, and many more. This document is about how to get started.

## What are these large language models?

Most (LLMs) today, including the GPT family of models from OpenAI, are word generators. You give them a prefix, and they keep generating tokens until they hit a limit or a pre-defined stop token. If I present the model with a prefix, it just keeps writing, almost like a test question.

Because they are trained on huge volumes of text, they know about a huge variety of things from the real world. To Microsoft AI: Artificial Intelligence Basics demonstrate, I’ll show a prompt or prefix in normal text, followed by the GPT-3 prediction in green highlights. It has learned about internal combustion engines:

Text

Description automatically generated

As well as natural language processing techniques:

Why has NLP shifted to use neural network models for langauge understanding?
NLP has shifted to use neural network models for language understanding due to the immense success of deep learning models in a variety of tasks. Neural network models are very powerful and effective in capturing the contextual information from an input text. They are also able to learn complex patterns and features from large amounts of data, making them well-suited for language understanding tasks.

The fluency and background here is impressive, though one can imagine how these completions and responses could be learned directly from the web – perhaps this answer or something like it was part of the training data, and then model has memorized this information.

More impressively, though, these models have also learned to work through information based on background context. For instance, they can solve word problems:

Text

Description automatically generated with medium confidence

The capacity of these models to capture common sense information, world knowledge, and reason over local information in text has dramatically increased over the past 5 years, as the training data provided to the model has grown and the computational capacity and parameters of the models have increased.

## How has this changed machine learning?

A few years ago, if we wanted a system to do complex pattern recognition or prediction, we would need to find large volumes of training data. Mostly this would mean thousands, or millions, or even billions of input/output pairs used to train a predictor. Translation systems would learn from pre-translated text, image recognition systems would learn from annotated images, etc. However, each system was unique and bespoke. This processing of ***training a machine learned model*** has been a key recipe to successful ML approaches starting in the 1990s.

* About 5 years ago, our approach to machine learned tasks changed drastically with large, pre-trained models like ImageNet and BERT, that learned background information about images or language. Machine learned approaches would start with one of these pre-trained models, and then ***fine-tune the model parameters*** (make smaller updates from an initial model starting point) to make the desired prediction. This allowed us to use orders of magnitude less training data and to have models with a broader coverage of background knowledge.

The most recent advance has been to move to so-called ***few-shot or zero-shot training***. Few-shot means that the model doesn’t need thousands or even hundreds of training examples anymore – just a handful of examples are used. Sometimes the model doesn’t even need a training instance anymore – zero-shot means that we just ask the model to do what we want, and it does it.

Graphical user interface, text, application

Description automatically generated

This prompt starts with an instruction, and then shows several examples of what is expected. Finally, the desired test input is provided. GPT successfully predicts a good completion. This seems like a bit of magic, but keep in mind that GPT was trained to mimic what people have written. The internet is full of documents where questions are answered, where reasoning is slowly worked out, where the chain-of-thought behind an answer is spelled out, and GPT has learned a lot from these cases.

Often people will call the input to these models a ***prompt***. Finding a good prompt that maximizes accuracy from large language model is called ***prompt engineering***. Developing good prompts is not a clear science yet – currently people manually tune the text (including written instructions, background context, provided examples components, etc.) to optimize the quality of output. There are efforts to make prompt engineering into a standard optimization problem, but this is in flux. Furthermore, there are advantages to written prompts – they can be easily tuned or updated without knowing how to code or optimize.

## What is still tough?

Although the landscape has drastically changed, some key ML problems have not disappeared, and other new issues have arisen. I’m listing three issues here that are important, but this list is not comprehensive.

### Evaluation

One key challenge in machine learned systems is ***evaluation*** – did my model do a good job? Does it work well in general? Will it be okay for the next piece of data or user who comes along? This challenge still remains. Often times, we’ll try out a few examples of GPT and find they work beautifully. However, sometimes they fail miserably.

Most machine learned systems are evaluated on a test set. For many tasks, finding and building a good test set isn’t easy. Sometimes we can use naturally occurring data for test (and maybe training too). For instance, human translators are paid to translate software documentation sentence by sentence. The resulting sentence pairs can be used to train and evaluate a system – if we’re lucky, such data will be available, and we can compare system outputs to a human-produced answer. If we’re not so lucky, we’ll need to create or curate that dataset. This is very important to ensuring that a system works well.

Furthermore, in many language-oriented examples, the system output and the human authored output may use totally different words but mean the same thing! For instance the two sentences “I have a severe headache” and “my head is throbbing with pain” have zero words in common, but are very close in meaning. The language processing community has developed noisy but useful techniques to compare system outputs, but this is imperfect.

Keep in mind that LLMs have transformed the process of developing and exploring a model, but clean and believable evaluation is a crucial and expensive part of an effective deployed system.

### Factual correctness

A major headache is that these large language models generate very authentic looking prose (very few spelling or grammatical errors), but often time make up information that is not grounded in the real world. I’ll ask GPT about myself:

Table

Description automatically generated

None of this is true. I’m not a football player or coach. Joe Brady is the offensive coordinator for the Panthers. The paragraph sounds very impressive – generating this kind of background pedigree might even get you elected to congress – but it’s not true.

Finding ways to identify and minimize this hallucination is a key problem area. Any downstream user needs to be cautious that the information from the model may be incorrect.

### Responsible AI

These models have learned many things from our training data, some desirable, some not. Amongst the model training data are buried explicitly problematic language (statements that are racist, etc.), as well as implicit biases (perhaps there are more occurrences of “he” associated with “doctor”, leading the model toward biases that humans make). When we use such a model to generate text or to guide decision making processes, it can potentially cause or amplify harm.

I cannot do justice to this crucial and complex issue in a getting started guide, but I ***strongly encourage*** users of any advanced pattern recognition or machine learning technique (especially powerful language models like these) to carefully consider the implications, improvements, and potential harms of their application. One starting point is here: <https://microsoft.sharepoint.com/teams/exdrai>

# Let’s build something

To pick a common task, say I want to summarize some data. I have a long body of text, and I’d like to find a shorter form of it. I’ll work with the Foundry Toolkit in these examples.

## Essential guidance for FHL and beyond

Our obligations to safeguard Microsoft and customer data don’t change even when doing FHL projects. Please keep the following in mind as you embark on your FHL awesomeness.

While using external endpoints like ChatGPT hosted at openai.com (or any other non M365 endpoint):

* Do not send any employee personal data to external endpoints. This includes data from internal tools, data from your own usage of Microsoft products, your own productivity data (e.g. your emails), or even data voluntarily provided by other employees.
* Do not send any confidential Microsoft information to external endpoints.
* Do not send any data that identifies customers (OII) or users (EUII/EUPI) or that has customer content to external endpoints.
* Only non-personal data, web data or synthetic data can be used against external endpoints.

There are internal endpoints like Foundry that are set up for experimentation. While using internal endpoints, keep the following guidelines in mind:

* Never send highly confidential or tented information, even to these internal endpoints.
* Use approved methods like [Recife data](https://docs.substrate.microsoft.net/docs/Substrate-Intelligence/Compliant-experimentation/Substrate-datasets/Substrate-data-sources/Recife.html?uid=eyes-on-recife&toc=%2Fdocs%2FSubstrate-Intelligence%2Ftoc.html) or using [Graph Explorer](https://developer.microsoft.com/en-us/graph/graph-explorer) for getting your personal data.
* The eyes-on or eyes-off guidance around specific types of data still holds true even when using internal endpoints. Please refer to the [data classification taxonomy](https://aka.ms/EntOnlineSvcDataTaxonomy) and the [data handling standards](https://aka.ms/o365datahandling) for guidance around data handling policies.

If you have any questions or need clarifications, please reach out to your compliance, security or privacy champion.

## Simple prompt

We are going to create first a prompt that asks GPT to summarize. You can jump to a link here:

<https://foundrytoolkit.azurewebsites.net/playgroundv2?session=7f34caca-ec5a-452f-bb24-3c4d6fc2449c>

The input to the model is as follows:

Summarize below text in 50 words:

I've been doing these pitch sessions for a year or two now and and this team and I always try to keep this like relatively informal, because next week is not only our FHL week, but it's also sort of the company wide hackathon and there should be an opportunity for people to do like crazy, interesting, unusual stuff to explore some new areas to really learn and hack and play around with things I guess one bit of feedback. I was just hearing today is that fix is a little bit tricky, right? Because fixes can sort of. Like feel a little bit doing your daily job just for an additional week. Of course, you're welcome to do that, but this should be a week that you have to use. However, you will to explore something completely different to just learn to create a new prototype of something. And so you and your whole management change should be getting encouraged to use this week in the way that you like. So if you have any feedback or issue about that, I do encourage you to take it up with your manager and so on. But I think the the feedback I generally hear across this org is everybody's excited about this, that.

That’s it – we just ask this model to summarize a given text. No training data, no fancy structure, no complex call structure – just textual input, and textual output

When provided with that input, GPT produces the following output:

The speaker has been doing pitch sessions for a year or two, and they are trying to keep it informal. They want people to use the hackathon week to explore something new and different, and not just use it to do their regular job.

It’s a pretty reasonable summary. You can play with the details too:

* Ask it to summarize in 10 words
* Vary the input text
* Explore variations in the instruction

It’s not always perfect, but keep in mind: this is not a model trained specifically to summarize. Rather, it has learned to consume text and follow generalized instructions. It does a pretty remarkable job.

Adam Troy has developed a [Prompt Playbook](https://microsoft-my.sharepoint-df.com/:w:/p/adtroy/Ecrwhj8FEo9Iqrq8wRi-JmoBH9uu2Yo6g3OVT1w5Nb__dA?e=bMrwrC) to define and illustrate some of the most effective approaches in prompt engineering today.

## Prompt template

Now if we actually want to use this in practice, we most likely want a more generalized form of template – something where the text to summarize, and maybe even the length of the summary, are parameters that can be specified at runtime.

The approach that Foundry Toolkit has used is to introduce a templating strategy. In addition to fixed text, a ***prompt template*** can contain placeholders that are replaced on the fly. The above prompt can be rewritten as (<https://foundrytoolkit.azurewebsites.net/playgroundv2?prompt=177ba41d-7758-4873-b8d4-68f4db469df6>):

Summarize below text in 50 words:

{{Text}}

The value of the {{Text}} variable can be supplied at runtime. Foundry Toolkit uses JSON to provide these parameter values and the [Handlebars language](https://handlebarsjs.com/) for templating. Other systems might use a different templating format or other variations. However, the notion of ***composing an input on the fly for GPT to operate over*** is powerful and commonly used these days.

It’s also worth noting that GPT does not (yet!) know about these template languages. This is just preprocessing that a user of GPT does beforehand.

This is a good point to reemphasize the importance of evaluation. One could try lots of variations on the input – put the instruction after the text, changing the wording of the instruction itself, adding some delimiter about the text itself, providing some additional input/output examples as part of the prompt, etc. But would this improve the quality of summaries overall? To answer this sort of question, you’ll need some kind of evaluation methodology and metrics, for instance a batch of known-good input/output pairs and some metric to compare human-authored and system-produced summaries.

## Calling the LLM API from Code

A critical part to accessing LLM endpoints is the LLM API ([aka.ms/llm-api-docs](https://aka.ms/llm-api-docs)). The LLM API is an abstraction layer which makes the compliant GPTx endpoints available to scenarios using appropriate authentication and manages the scope of targeting to users and providing rate limiting leveraging Azure OAI.

The repo at [aka.ms/lm-api-examples](https://aka.ms/llm-api-examples) contains some code to get you started, including simple examples in Python and C# calling into an LLM API endpoint. (More examples will be coming soon!)

You can use your own Recife personal curated email data as prompts to get completions results from the OpenAI Endpoint. We have created a code-lab with a hands-on example on how to do just that: [aka.ms/llm-studio-example](http://aka.ms/llm-studio-example).

# Example use cases

There are many possible ways to explore GPT models. Here are just a few ideas:

* **Asking about your calendar.** You can get a snapshot of your calendar from Microsoft Graph, serialize it out into text, then include it as part of a prompt for asking about past and future events. Finding effective means of presenting our Substrate data as part of a prompt is a key question for MSAI and Microsoft in the future.
* **Authoring documentation automatically.** You can provide code and an outline about the goals of your document, then use GPT to help you construct the details of that documentation. Recent GPT models are surprisingly good at understanding code!  
  In line with this MSAI has joined the E+D wide Documentation Enhancement Program (DEP, [aka.ms/msai-dep](https://aka.ms/msai-dep)). Join forces using [aka.ms/msai-dep/fhl/join](https://aka.ms/msai-dep/fhl/join) to make your prototype part of the MSAI DEP.
* **Generate meeting agendas, based on meeting titles and attendees.** Leveraging information from your user shard to gather information about attendees and background about a meeting title, you could explore authoring an agenda. This could be informed by past meetings with similar titles, information or topics associated with the people on the invite, and relevant documents. Gathering the data and establishing an evaluation could be an interesting challenge.
* **Generate diagrams from text in some simple markup language (e.g.** [**mermaid**](https://mermaid-js.github.io/docs/mermaid-live-editor-beta/#/edit/eyJjb2RlIjoiZ3JhcGggVERcbkFbQ2hyaXN0bWFzXSAtLT58R2V0IG1vbmV5fCBCKEdvIHNob3BwaW5nKVxuQiAtLT4gQ3tMZXQgbWUgdGhpbmt9XG5DIC0tPnxPbmV8IERbTGFwdG9wXVxuQyAtLT58VHdvfCBFW2lQaG9uZV1cbkMgLS0-fFRocmVlfCBGW2ZhOmZhLWNhciBDYXJdXG4gICIsIm1lcm1haWQiOnsidGhlbWUiOiJkZWZhdWx0In0sInVwZGF0ZUVkaXRvciI6ZmFsc2V9)**).** Sometimes a picture really is worth a thousand words. Can you generate a flowchart or system diagram automatically given a piece of text?

Text Generation

* Produce documentation from design specs, code, etc. (things that go into SDC)
* Write TSG based on set of Teams, Stack Overflow questions and documentation.
* Automate QOS, COGS, sprint or status report generation.
* Create a text to YAML for compliant AML components or Polymer configurations using ChatGPT and custom VSCode plugins. Similar could be done for Polymer configurations (e.g. protobuf and other configurations). *(POC:* [*Amund.Tveit@microsoft.com*](mailto:Amund.Tveit@microsoft.com)*)*
* Agenda creation: Train ChatGPT to generate agendas for meetings based on the topics, attendees, and goals of the meeting.
* Customized image generation: e.g. suggest slide layout/images but based on context and user’s past slide show content

Summarization

* Q&A over and summarize a document.
* Question generation off a document to validate reading comprehension
* Q&A over my inbox
* Ask questions about my calendar: when am I talking to X next?
* Generate highlights/lowlights/risks from engineering reports or ADO status.
* Catch up with a chat or email thread you haven’t read
* Summaries of stack overflow questions on a given topic
* Generate a summary of what my meetings (tomorrow/next week) are about by leveraging calendar, people, roles, docs, LinkedIn, who+...

Search

* Search for SDC docs using ChatGPT on SDC content
* Helping users find something relevant if their search has failed by multi-turn clarification of your query. The goal of the latter thing would be to get a trace of a search that would have failed, but now succeeded, and this could be used to improve our system.

Internal tooling

* Pipelines connecting semantic fabric or SWG with LLMs: e.g. prompt augmentation or fetching data for improved ChatGPT responses
* Using (Chat)GPT to detect hallucinations/textual entailment through coaching. Given original text and generated text (could be from GPT or handcrafted), can we provide the necessary information in the prompts to determine whether the generated text is grounded in the original content.
* Develop a full-fledged GPT-Annotator: pick a task like email classification, leverage GPT to generate an Annotation Guideline, use the Annotation Guideline as input and ask GPT to generate prompts for it to do synthetic +ve and -ve sample generation. Use GPT and the prompts from previous step to produce positive and negative examples. Assess via human judgement whether the synthetic dataset seems useful/consistent
* Create a completely simulated environment for compliant AML and/or Polymer using ChatGPT similar to [Linux terminal](https://www.reddit.com/r/linux/comments/ze2pg8/chatgpt_knows_linux_so_well_you_can_emulate_it/) or Microsoft Graph simulations *(see slide 35-38 in this deck:* [ChatGPT Usage Examples.pptx](https://microsoft-my.sharepoint-df.com/:p:/p/amtveit/EVjtlSKyugZLlSUBlKZsWBMBmaQ6MYyR5B0Uz5QsZ7xA3Q?e=PE9le1)) *(POC:* [*Amund.Tveit@microsoft.com*](mailto:Amund.Tveit@microsoft.com)*)*

Recommendation

* What should I spend my time doing next? What documents I should read?
* Custom email triage: move emails to correct folders customized to users’ directory structure (I.e. get rid of manual email rules)
* Event recommendations: Train ChatGPT to recommend events to users based on their previous attendance, interests, and location.
* Recommendation of [movies using Movie Lens or Teams Chats or Outlook Emails] via GPT prompts that includes instructions (inject Substrate data for Teams/Outlook), actual question, negative and positive examples. Chat/Email – is this an important chat/email to focus on?

# Support and resources

You can use the following support channels to get assistance using our platform.

* [Ask in Stack Overflow](https://docs.substrate.microsoft.net/docs/Substrate-Intelligence/Compliant-experimentation/Substrate-AI-support/Stack-Overflow.html): For general questions about the platform, how to use it, etc. Please use the ***substrate-llm-suppor*t** tag in your question.
* [Report an Issue](https://docs.substrate.microsoft.net/docs/Substrate-Intelligence/Compliant-experimentation/Substrate-AI-support/Reporting-an-issue.html): For agility blockers, general feedback, feature requests, and documentation change requests/feedback. Please use **s*ubstrate-llm-support*** tag while reporting your issue.
* Join [AI Platform Users](https://idwebelements.microsoft.com/GroupManagement.aspx?Group=aiplatformusers&Operation=join) DL to receive the latest Substrate Intelligence Platform updates including LLMs.

Other places to look for useful information:

* [E+D AI Accelerator - Home (sharepoint.com)](https://microsoft.sharepoint.com/teams/ExDAI)
* [OPG FHL Talks on LLMs](https://microsoft.sharepoint.com/teams/ExDAI/SitePages/exdfhl.aspx?xsdata=&sdata=dnNOWFlsamdzYitBTmZKVVcyRkFVejFsaWsrdWluK1NPTjcwQlhMSndUQT0%3D&ovuser=72f988bf-86f1-41af-91ab-2d7cd011db47%2Cchrisq%40microsoft.com&OR=Teams-HL&CT=1675984315264&clickparams=eyJBcHBOYW1lIjoiVGVhbXMtRGVza3RvcCIsIkFwcFZlcnNpb24iOiIyNy8yMzAyMDUwMTQwMyIsIkhhc0ZlZGVyYXRlZFVzZXIiOmZhbHNlfQ%3D%3D)
* [Building GPT-3 applications — beyond the prompt](https://medium.com/data-science-at-microsoft/building-gpt-3-applications-beyond-the-prompt-504140835560) by Paulo Salem

# FAQs

**Which OpenAI models are supported by Substrate LLM API?**

Substrate LLM API supports 4 OpenAI models today:

* GPT-3: text-davinci-001
* GPT-3.5: text-davinci-002
* GPT-3.51: text-davinci-003
* ChatGPT: text-chat-davinci-002

The list of available models will be kept up-to-date at this link [aka.ms/inferencing/llm](https://aka.ms/inferencing/llm).

**How much traffic can I direct to Substrate LLM API during FHL?**

We are requesting teams to limit their usage of Substrate LLM API to prototyping scenarios during FHL (5-10 requests per hour per user). Teams that are interested in scaling their scenarios to bigger rings should reach out to Nitant Singh (nisi@) for next steps.

**I want to explore Large Language Models. How can I get started?**

Here are two tools you can use to explore Large Language Models:

* [Foundry Toolkit](https://foundrytoolkit.azurewebsites.net/playgroundv2)
* [Augmentation Loop Playground](https://web.augloop-tools.officeppe.com/playground/)

And of course, this guide!