**My road map in this project**

references :

https://www.youtube.com/@AndrejKarpathy

https://www.youtube.com/watch?v=zduSFxRajkE

https://github.com/brevdev/notebooks/blob/main/mistral-finetune-own-data.ipynb

https://medium.com/@thakermadhav/build-your-own-rag-with-mistral-7b-and-langchain-97d0c92fa146

https://www.langchain.com/

<https://huggingface.co/blog/how-to-generate>

Tokenization: <https://github.com/openai/openai-cookbook>

Fine-tuning: https: //[github.com/brevdev/notebooks/blob/main/mistral-finetune-own-data.ipynb](http://github.com/brevdev/notebooks/blob/main/mistral-finetune-own-data.ipynb)[first link]

Future: <https://github.com/ashishpatel26/LLM-Finetuning> It is advanced for fine-tuning after the first link

The below link is for Tokenization, especially for LLm:

<https://github.com/openai/openai-cookbook/blob/main/examples/How_to_count_tokens_with_tiktoken.ipynb>

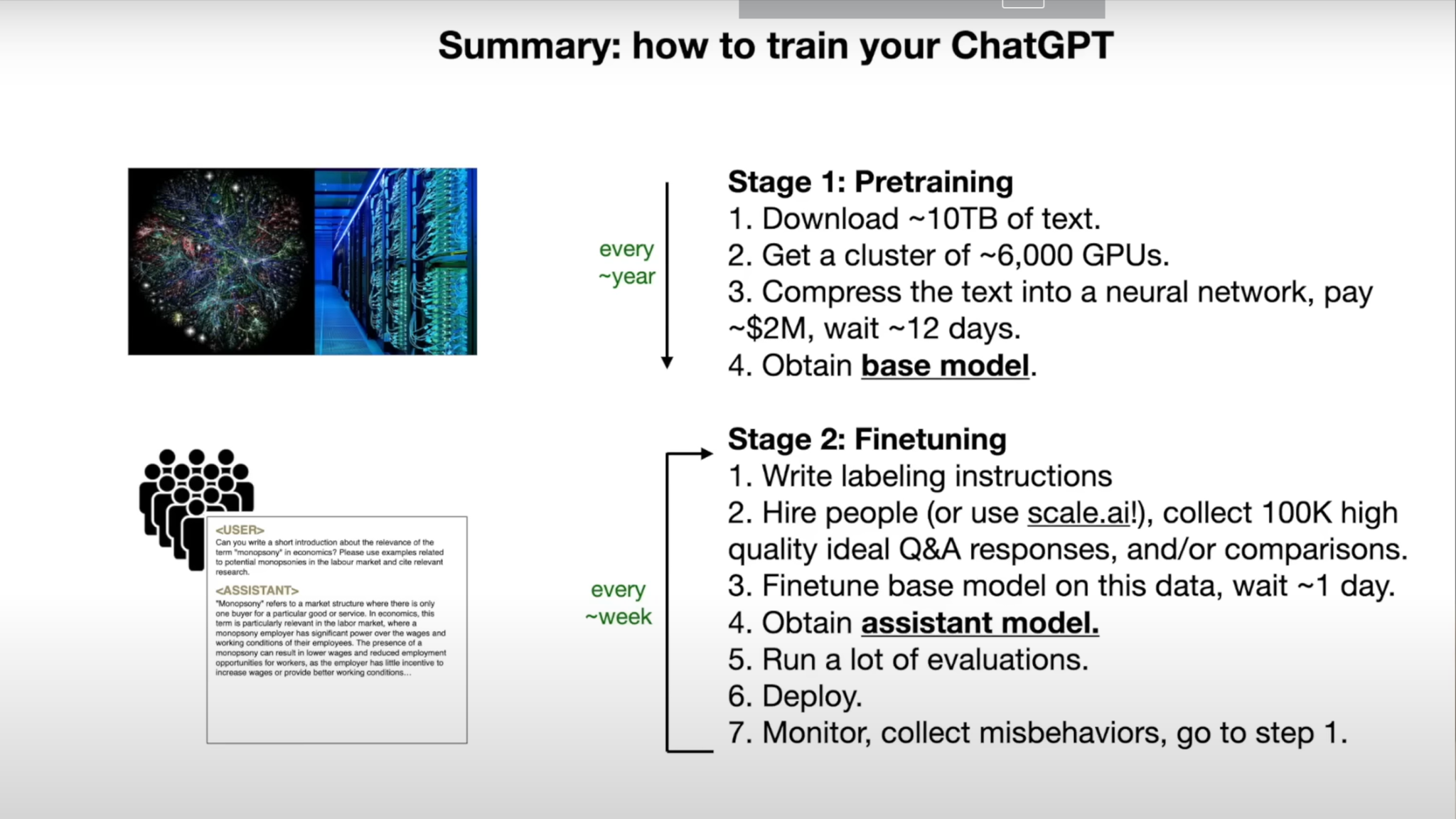
Self-supervised: <https://arxiv.org/pdf/2310.06825>

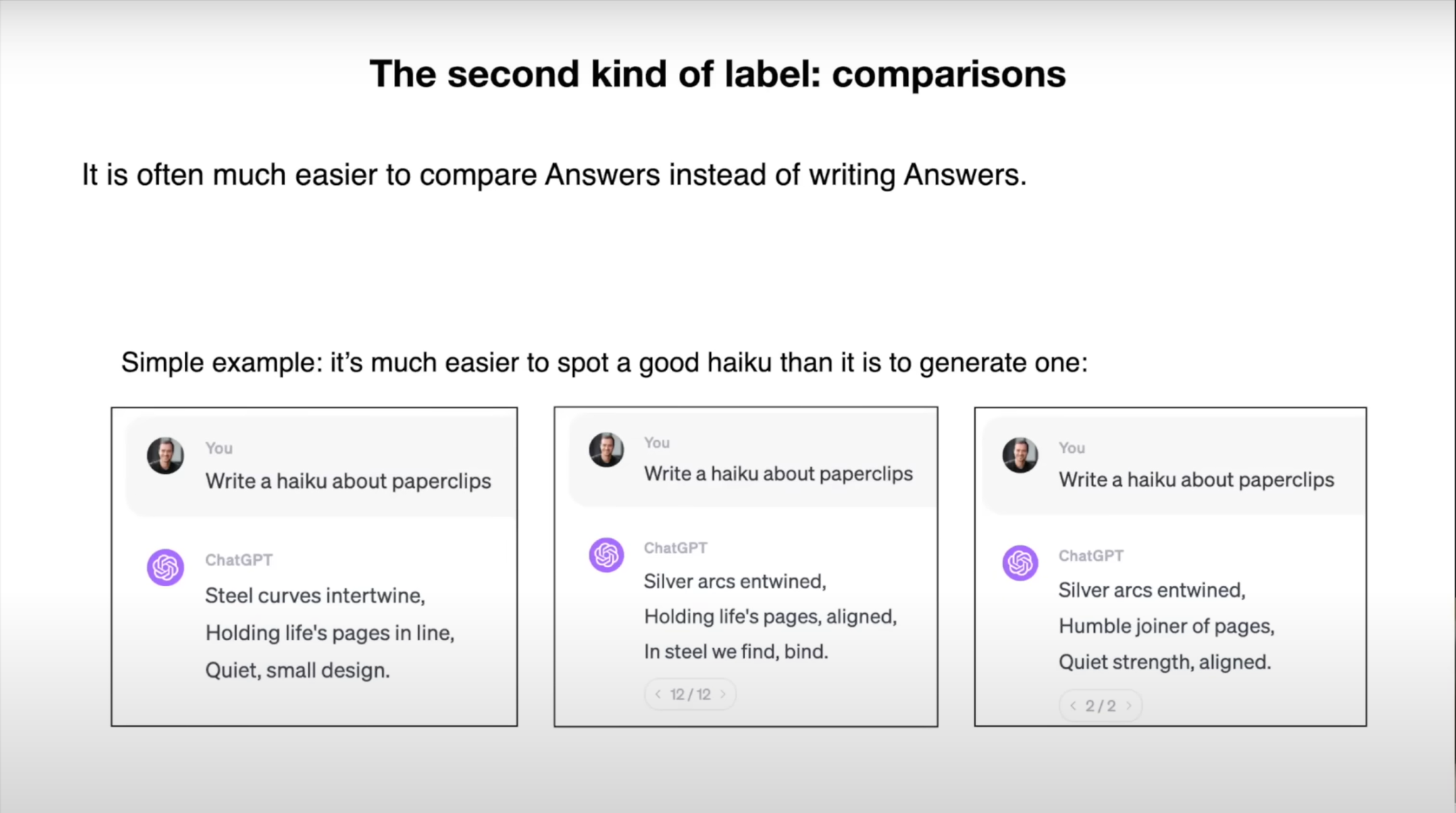
Maybe it is a survey for constraint decoding: <https://arxiv.org/pdf/2403.01632>

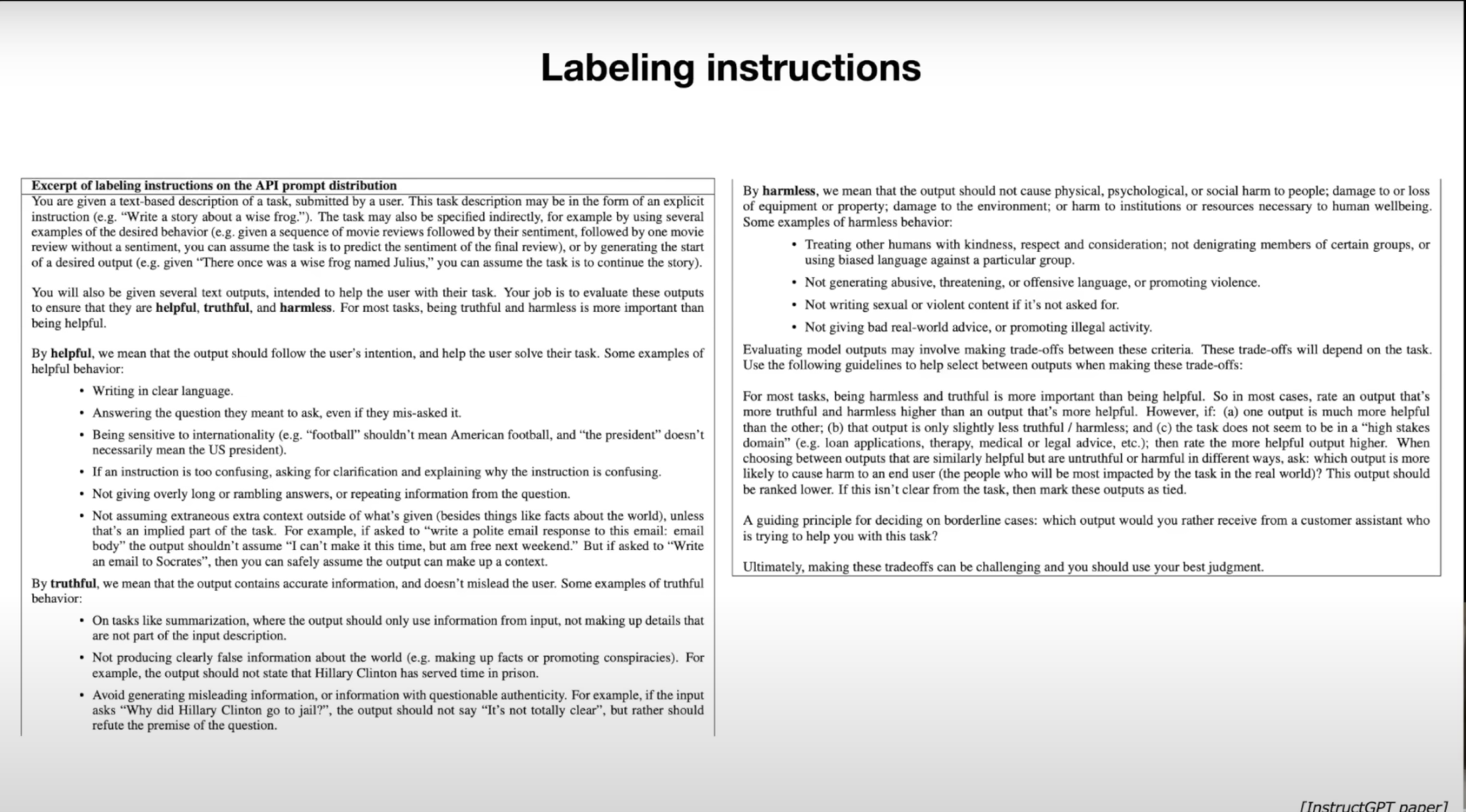
**Papers:**

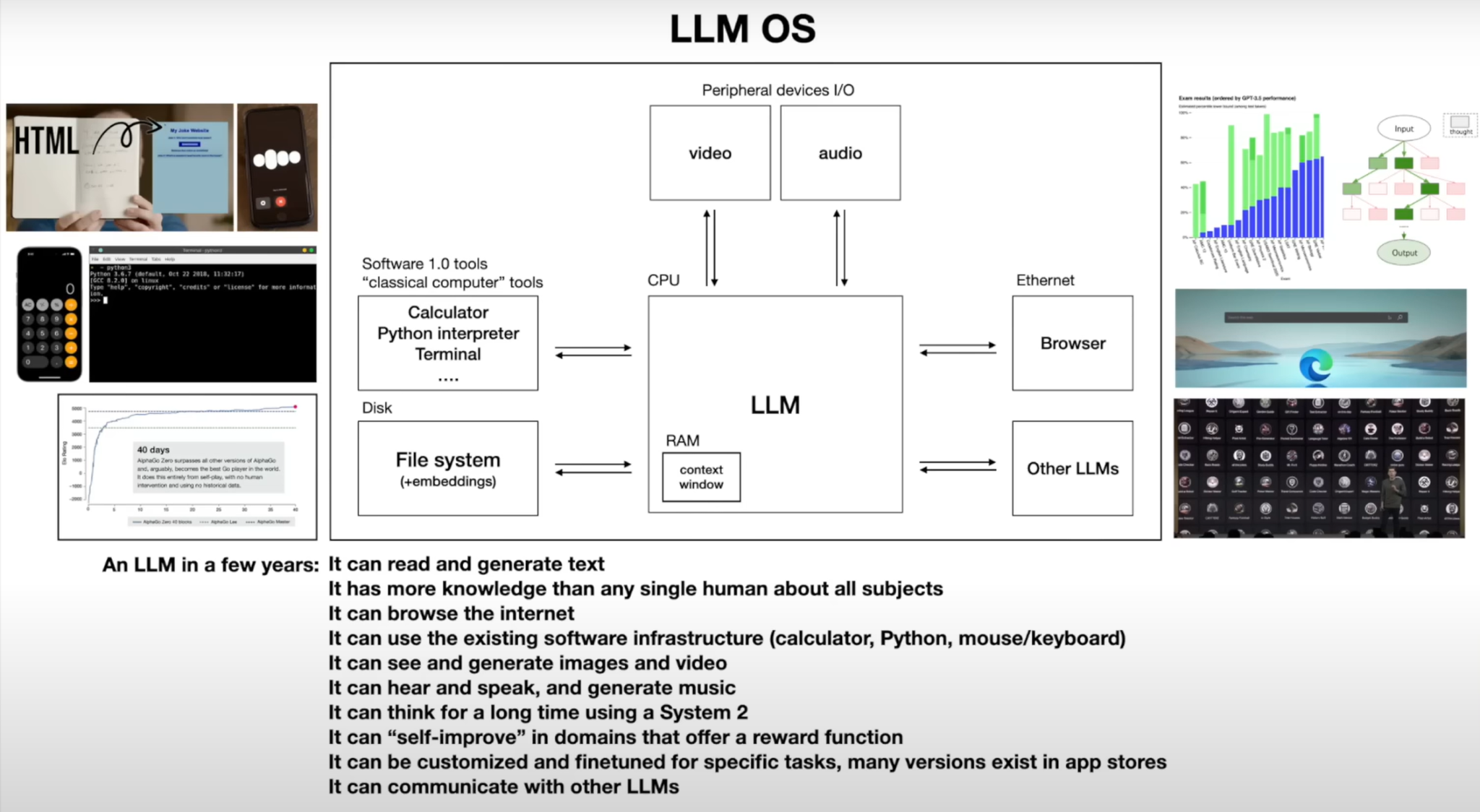
1. Grammar-constrained decoding for Structured NLP Tasks without fine-tuning
2. The Whole Truth and Nothing But the Truth: Faithful and Controllable Dialogue Response Generation with Dataflow Transduction and Constrained Decoding

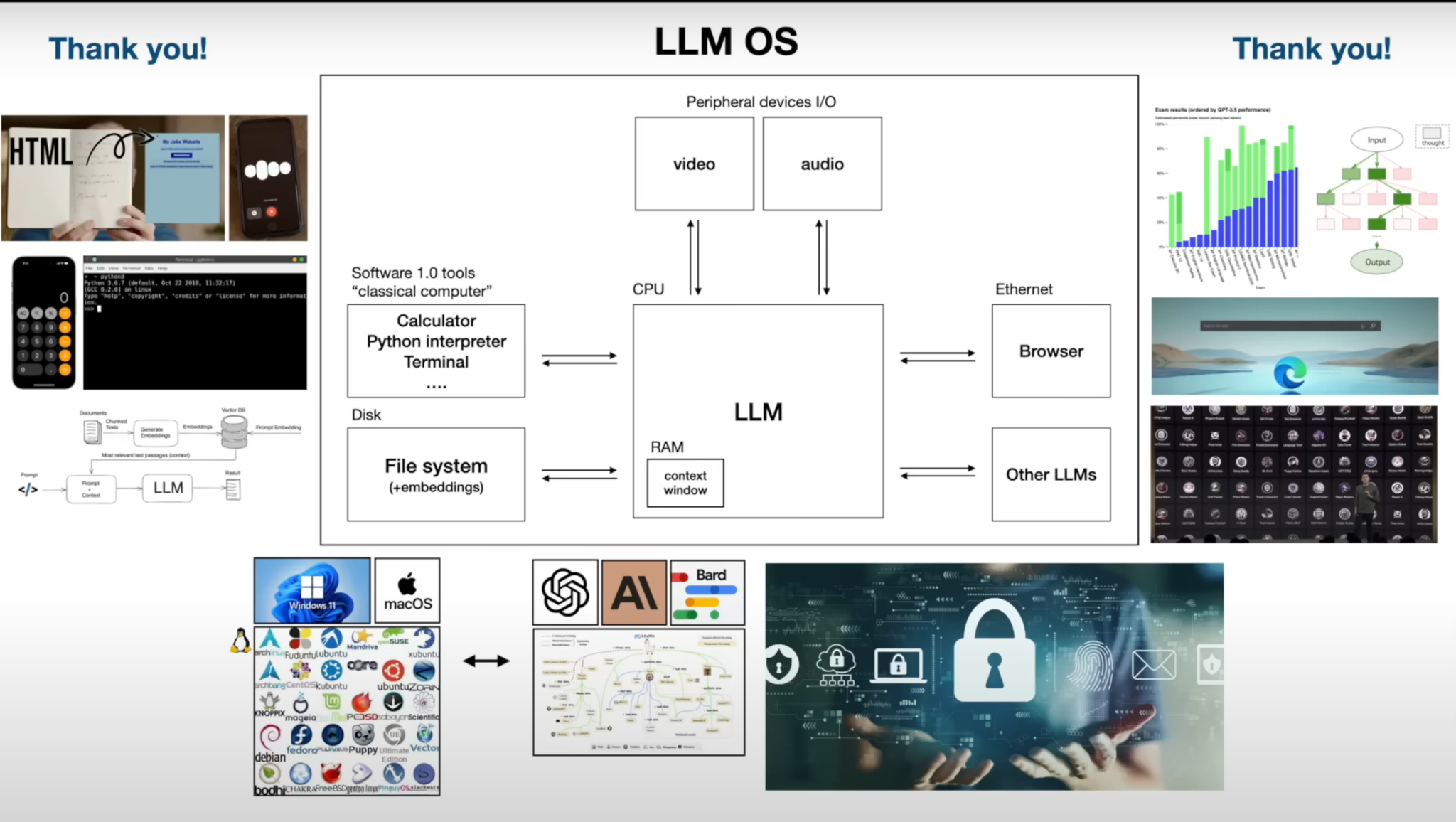
**First day of work on this project:**











My insight: This video was a short theory about LLM models. The presenter points to the key points of LLM.

The last part was exciting for me. It was about attacks in LLm models. I would like to know more about this.

**Task 2.** Read this paper [Grammar-Constrained Decoding for Structured NLP Tasks without Fine Tuning]

**Status:**

### **Introduction to the Paper**

The paper titled **"Grammar-Constrained Decoding for Structured NLP Tasks without Fine Tuning"** explores a method to improve how language models (like GPT-3) generate structured text.

At first, I was required to learn the main concept of this topic:

**What is Structured text?** is the text that follows a specific format or set of rules, such as filling out a form or generating code.

**Structured NLP Tasks:** These tasks require the output to follow a specific format.

**Ex of Structured NLP Tasks:**

* **Information Extraction:** Extracting specific information from text.
* **Entity Disambiguation:** Identifying the correct entity (like a person or place) mentioned in the text.
* **Constituency Parsing:** Analyzing the grammatical structure of a sentence.

**What is the Grammar-Constrained Decoding (GCD)** This method ensures that the text the AI generates follows a predefined structure or grammar. Think of it as giving the AI strict rules when generating text.

I need clear and understandable examples of grammar constraint decoding for better understanding.

**Ex of GCD:**

1. Closed Information Extraction (cIE) is a struchtured NLP task

Closed Information Extraction (cIE) involves extracting structured information, specifically subject-relation-object triplets, from unstructured text.

**Task:** Extract subject-relation-object triplets from the text.

**Input Text:** "Marie Curie discovered radium in 1898."

**Desired Output:** [subject: Marie Curie, relation: discovered, object: radium, year: 1898]

**Grammar Constraints:**

* Subjects must be valid entity names.
* Relations must be valid relation names.
* Objects must be valid entity names.
* Years must be valid years.

**Grammar Rules:**

* S -> [subject] [relation] [object] [year]
* [subject] -> "Marie Curie" | "Albert Einstein" | "Isaac Newton"
* [relation] -> "discovered" | "invented" | "proposed"
* [object] -> "radium" | "the theory of relativity" | "calculus"
* [year] -> "in 1898" | "in 1905" | "in 1687"

**Decoding Steps:**

1. **Generate Subject:**
   * Allowed Tokens: "Marie Curie", "Albert Einstein", "Isaac Newton"
   * Model Suggests: "Marie Curie"
   * Output: "Marie Curie"
2. **Generate Relation:**
   * Allowed Tokens: "discovered", "invented", "proposed"
   * Model Suggests: "discovered"
   * Output: "discovered"
3. **Generate Object:**
   * Allowed Tokens: "radium", "the theory of relativity", "calculus"
   * Model Suggests: "radium"
   * Output: "radium"
4. **Generate Year:**
   * Allowed Tokens: "in 1898", "in 1905", "in 1687"
   * Model Suggests: "in 1898"
   * Output: "in 1898"

**Final Output:**

* [subject: Marie Curie, relation: discovered, object: radium, year: 1898]

### **Why is GCD Important?**

* **Current Challenges:** LLMs are good at generating text but struggle with tasks requiring a specific structure without fine-tuning (additional training for specific tasks).
* **Solution with GCD:** Using formal grammar (rules), GCD can guide the AI to produce text that follows the required structure without needing extra training.

### **How does GCD work?**

1. **Formal Grammars:** These are sets of rules that define the output structure. For example, in information extraction, the output might need to be in the format "subject-verb-object" (e.g., "John-buys-apple").
2. **Input-Dependent Grammars:** These adjust the rules based on the input text, allowing the AI to generate different structures as needed.

### **What are the Benefits of GCD?**

* **Flexibility:** GCD can be used for various tasks without fine-tuning the AI for each task.
* **Performance:** In experiments, GCD-enhanced models performed better than those without constraints, even matching or surpassing some task-specific models.

#### **Example of GCD:**

#### **Problem:**

LLMs are very good at generating text but need additional training to generate text that follows specific structures or formats.

#### **Solution: Grammar-constrained decoding (GCD)**

In this paper, GCD is a method proposed to guide the text generation of LLMs by using formal grammar to ensure the output follows a required structure.

### **Main Contributions of the Paper:**

1. **Unified Framework:** The paper demonstrates that many structured NLP tasks can be framed as grammar-constrained decoding problems.
2. **Input-Dependent Grammars:** Introduces grammars that can adjust based on the input text, allowing more flexibility in generating different structures for different inputs.
3. **Empirical Demonstration:** Shows through experiments that GCD-enhanced LLMs perform significantly better in tasks like information extraction, entity disambiguation, and constituency parsing

I started with this link [<https://github.com/brevdev/notebooks/blob/main/mistral-finetune-own-data.ipynb>] I understood before fine-tuning I needed to learn Tokenization, so I continued with this link [ <https://github.com/openai/openai-cookbook>.]

**What is tokenization?**

Reference: “<https://huggingface.co/docs/transformers/en/preprocessing>”

Before you can train a model on a dataset, it needs to be preprocessed into the expected model input format. Whether your data is text, images, or audio, they need to be converted into tensors batches. In Huginface there is a library (Transformers: it provides a set of preprocessing classes to help prepare your data for the model.)

* Text uses a [Tokenizer](https://huggingface.co/docs/transformers/en/main_classes/tokenizer) to convert text into a sequence of tokens, create a numerical representation of the tokens, and assemble them into tensors. The main tool for preprocessing textual data is a [tokenizer](https://huggingface.co/docs/transformers/en/main_classes/tokenizer). A tokenizer splits text into *tokens* according to a set of rules. The tokens are converted into numbers and then tensors, which become the model inputs. Any additional inputs required by the model are added by the tokenizer.

**What is the** [**Tokenizer**](https://huggingface.co/docs/transformers/en/main_classes/tokenizer)**?**

The Hugging Face Tokenizer documentation explains how tokenizers prepare inputs for models, including tokenization, converting tokens to IDs, and encoding/decoding sequences. There are two implementations: a full Python version and a faster Rust-based version. Key classes include PreTrainedTokenizer and PreTrainedTokenizerFast, which manage tokenization methods, adding new tokens, handling special tokens, and more. It also details batch encoding, managing token attributes, and configuring tokenization options such as padding, truncation, and special tokens**.** Both classes inherit from PreTrainedTokenizerBase, which includes shared methods, and SpecialTokensMixin, which handles special tokens like [CLS], [SEP], etc.

**Fast tokenizer :**

* Significant Speed-up with Batched Tokenization: When batched tokenization is done, the PreTrainedTokenizerFast class significantly speeds up. This is because it leverages the fast implementation in Rust.
* Mapping Between Original String and Token Space: The PreTrainedTokenizerFast class offers methods to map between the original string (characters and words) and the token space

Example of a Fast tokenizer: https://github.com/monirmo97/LLM/blob/main/Tokenizer.ipynb

**Different types of tokenization?**

**Link of GitHub**: <https://github.com/monirmo97/LLM/blob/main/Different_Type_of_tokenization.ipynb>

1. Word Tokenization:

* Splits text into individual words.
* Simple and intuitive.
* Example: "Hello, world!" → ["Hello", ",", "world", "!"]

1. Subword Tokenization:

* Splits text into subwords or morphemes.
* Useful for handling out-of-vocabulary words and reducing vocabulary size.
* Techniques include Byte-Pair Encoding (BPE) and WordPiece.
* Example: "unhappiness" → ["un", "happiness"] or ["un", "##happy", "##ness"]

1. Character Tokenization:

* Splits text into individual characters.
* Useful for languages with a large number of unique characters.
* Example: "Hello" → ["H", "e", "l", "l", "o"]

1. Sentence Tokenization:

* Splits text into individual sentences.
* Useful for tasks involving sentence-level processing.
* Example: "Hello world. How are you?" → ["Hello world.", "How are you?"]

**Tokenization in LLM?**

**……**

**What is the library for tokenization?**

**…….**

**Example of tokenization:**

**……..**

**How Can give input to LLM?**

**…..**

**Example of giving input**

**…….**

**What is prompt engineering?**

**……..**

**Example of Prompt**

**……..**

**Fine-tuning LLM**

**……….**

**LLM input:**

## **How to control a large language model?**

Of all the inputs to a large language model, by far the most influential is the text prompt.

Large language models can be prompted to produce output in a few ways:

* Instruction: Tell the model what you want
* Completion: Induce the model to complete the beginning of what you want
* Scenario: Give the model a situation to play out
* Demonstration: Show the model what you want, with either:
  + A few examples in the prompt
  + Many hundreds or thousands of examples in a fine-tuning training dataset

An example of each is shown below.

### **Instruction prompts**

Write your instruction at the top of the prompt (or at the bottom, or both), and the model will do its best to follow the instructions and then stop. Instructions can be detailed, so feel free to write a paragraph explicitly detailing the output you want, just stay aware of how many tokens the model can process.

1. **instruction prompt**

Extract the name of the author from the quotation below.

Ex of instruction prompt:

[

“Some humans theorize that intelligent species go extinct before they can expand into outer space. If they're correct, then the night sky's hush is the graveyard's silence.”

Ted Chiang, Exhalation

]

Output: Ted Chiang

### **Completion prompt:**

Completion-style prompts take advantage of how large language models try to write text they think will likely come next. To steer the model, try beginning a pattern or sentence that will be completed by the output you want to see. This mode of steering large language models can take more care and experimentation relative to direct instructions. In addition, the models won't know where to stop, so you will often need stop sequences or post-processing to cut off text generated beyond the desired output.

Ex completion prompt:

[

“Some humans theorize that intelligent species go extinct before they can expand into outer space. If they're correct, then the night sky's hush is the graveyard's silence.”

Ted Chiang, Exhalation

The author of this quote is

]

Output: Ted Chiang

### **Scenario prompt**

Giving the model a scenario to follow or a role to play out can be helpful for complex queries or when seeking imaginative responses. When using a hypothetical prompt, you set up a situation, problem, or story, and then ask the model to respond as if it were a character in that scenario or an expert on the topic.

Ex scenario prompt:

[

Your role is to extract the name of the author from any given text

“Some humans theorize that intelligent species go extinct before they can expand into outer space. If they're correct, then the night sky's hush is the graveyard's silence.”

Ted Chiang, Exhalation

]

Output: Ted Chiang

### **Demonstration prompt (few-shot learning)**

Like completion-style prompts, demonstrations can show the model what you want it to do. This approach is sometimes called few-shot learning, as the model learns from a few examples in the prompt.

Ex-demonstration prompt:

[

Quote:

“When the reasoning mind is forced to confront the impossible repeatedly, it has no choice but to adapt.”

N.K. Jemisin, The Fifth Season

Author: N.K. Jemisin

Quote:

“Some humans theorize that intelligent species go extinct before they can expand into outer space. If they're correct, then the night sky's hush is the graveyard's silence.”

Ted Chiang, Exhalation

Author:

]

Output: Ted Chiang

### **Fine-tuned prompt**

With enough training examples, you can fine-tune a custom model. In this case, instructions become unnecessary, as the model can learn the task from the training data provided. However, it can be helpful to include separator sequences (e.g., -> or ### or any string that doesn't commonly appear in your inputs) to tell the model when the prompt has ended and the output should begin. With separator sequences, the model can continue elaborating on the input text rather than starting on the answer you want to see.

1. Ex fine-tuned prompt (for a model that has been custom-trained on similar prompt-completion pairs):

[

“Some humans theorize that intelligent species go extinct before they can expand into outer space. If they're correct, then the night sky's hush is the graveyard's silence.”

Ted Chiang, Exhalation

]

Output: Ted Chiang

In addition to text prompts, there are other ways to input data into a large language model:

1. **Multimodal Inputs**: Combining text with images or other types of media (e.g., CLIP models by OpenAI).
2. **Structured Data**: Feeding in tables, graphs, or JSON objects.
3. **Interactive Systems**: Integrating with applications that provide dynamic, real-time inputs, such as chatbots or virtual assistants.
4. **Sensor Data**: Using IoT devices to send real-time sensor data as inputs.
5. **Voice Input**: Converting speech to text using speech recognition systems.

References[<https://github.com/openai/openai-cookbook>]