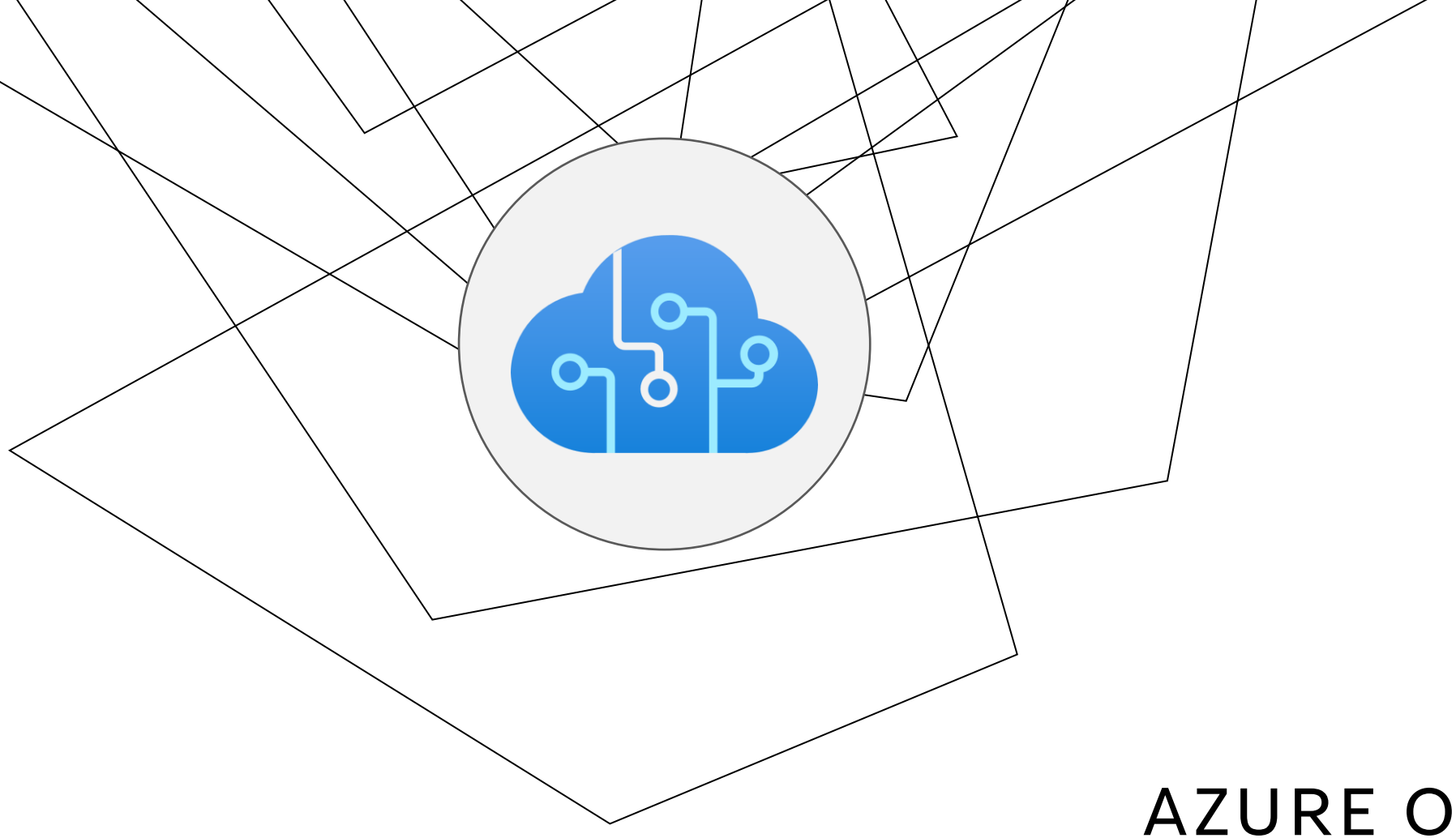
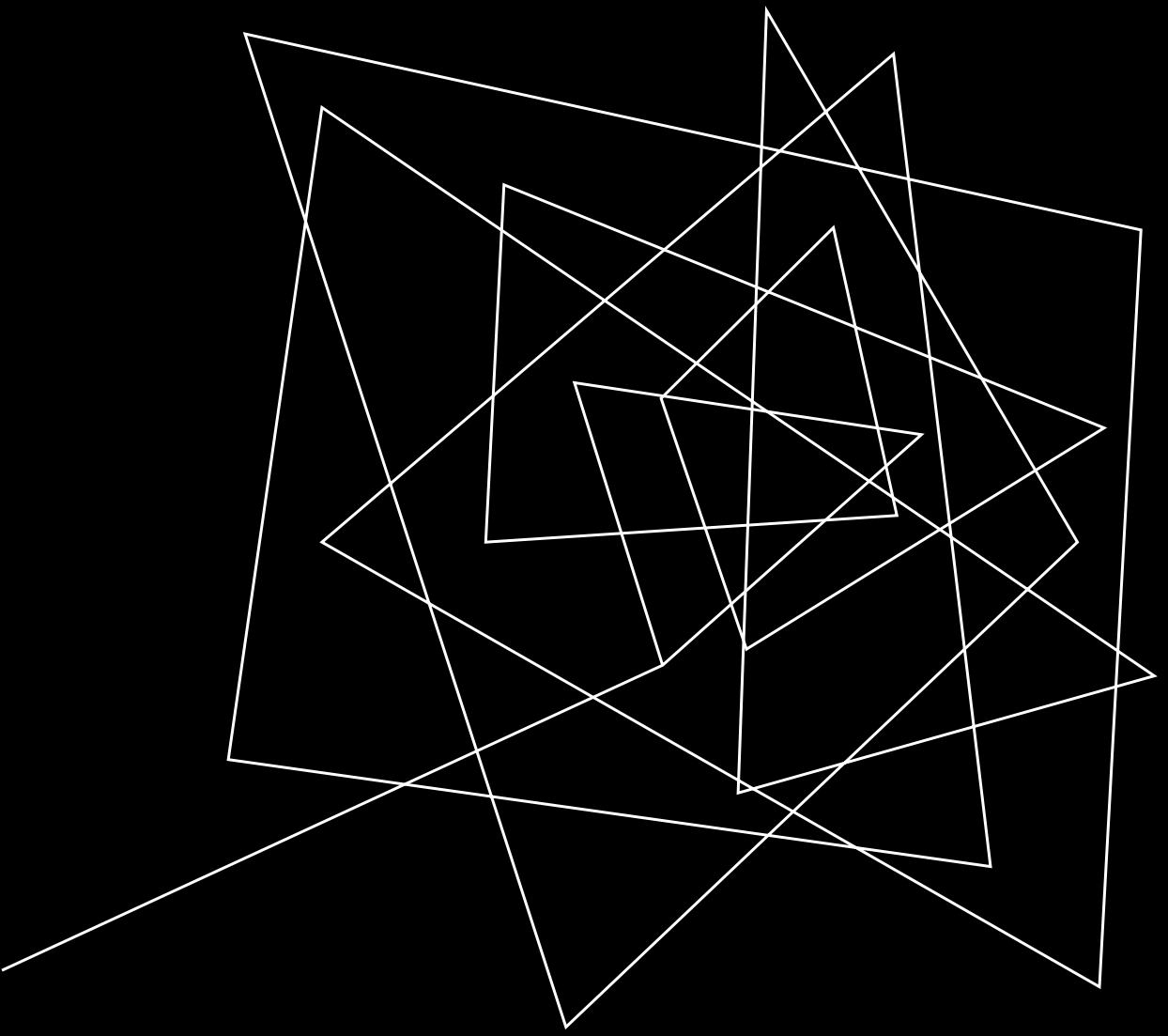


# AZURE OPENAI SERVICES

Overview



# MODELS OVERVIEW

MODELS OVERVIEW

# LLM TRAINING

**Process**

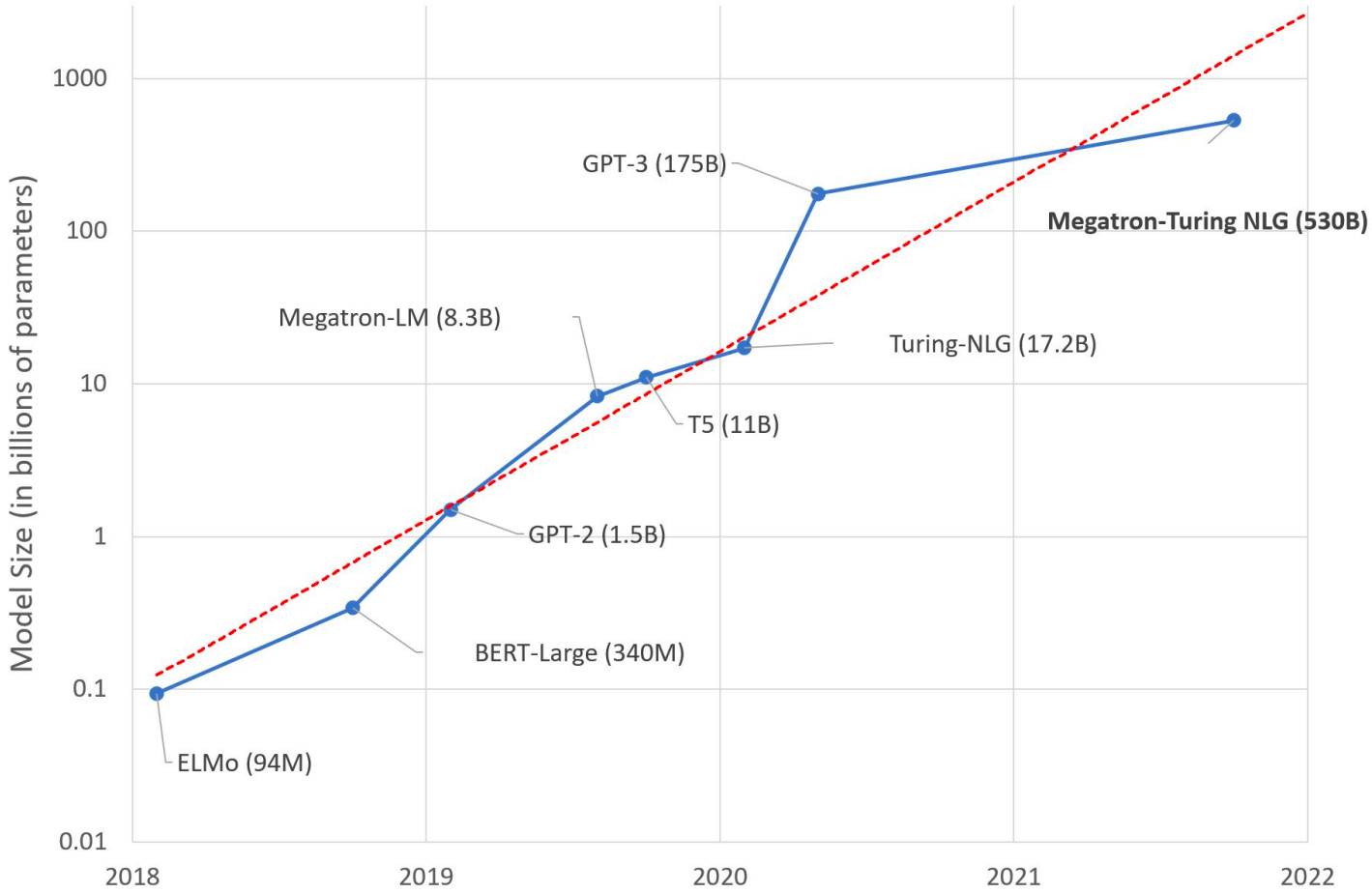
- Semi-Supervised Learning
  - Task: Next token prediction
  - Enables massive scale
- Reinforcement Learning through Human Feedback

**Data**

- Common Crawl – 60%
- WebText – 22%
- Books1 – 8%
- Books2 – 8%
- Wikipedia – 3%
- GitHub – Added for code models

MODELS OVERVIEW

# EXPLOSIVE GROWTH



MODELS OVERVIEW

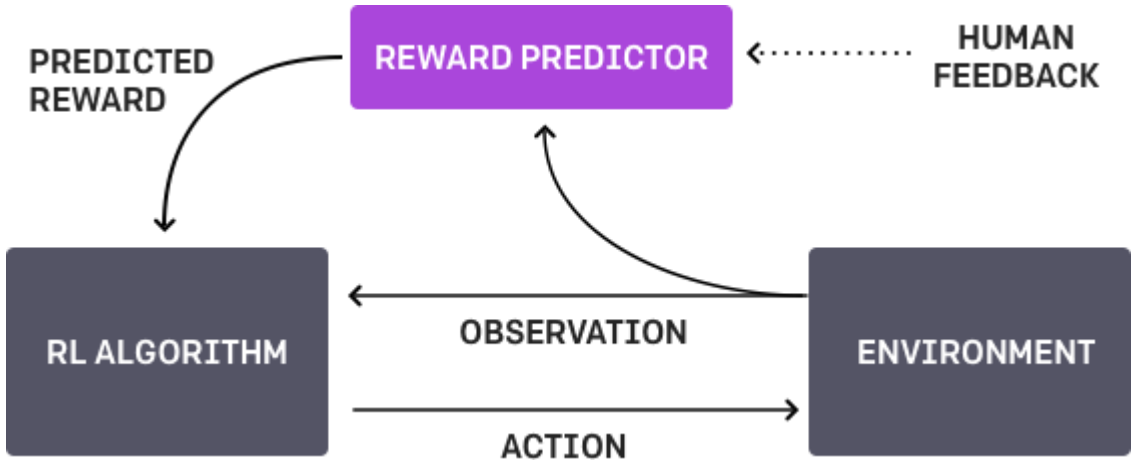
# GPT-3

- GPT-3 is a large-scale, advanced language model pre-trained on a large text corpus to predict the next word
- Key Result:
  - Unique capabilities with few-shot learning and generalizability, so GPT-3 can be useful for most natural tasks, without extra training for specific tasks.
- Core Capabilities
  - Generation (i.e., full sentences, cohesive paragraphs)
  - Classification
  - Transformation (i.e., summarization, translation, emoji)
  - Completion (i.e., generate React components)
  - Factual Responses (i.e., Q&A)
  - Conversation (i.e., chatbots)

MODELS OVERVIEW

# OPENAI TRAINING MODEL

- Code Training → Codex
- Reinforcement Learning from Human Feedback (RLHF)
  - Instruction tuning → GPT-3.5
  - Dialogue tuning → ChatGPT



MODELS OVERVIEW

# INTRO TO PROMPTING

- What is a prompt?
  - NL Perspective: “Priming” the model
  - ML Perspective: “Configuring” the model
- Parameters
- Tokens
- Concepts
- Challenges

https://aka.ms/foundry/toolkit

MODELS OVERVIEW

# MODEL NAME CHEAT SHEET

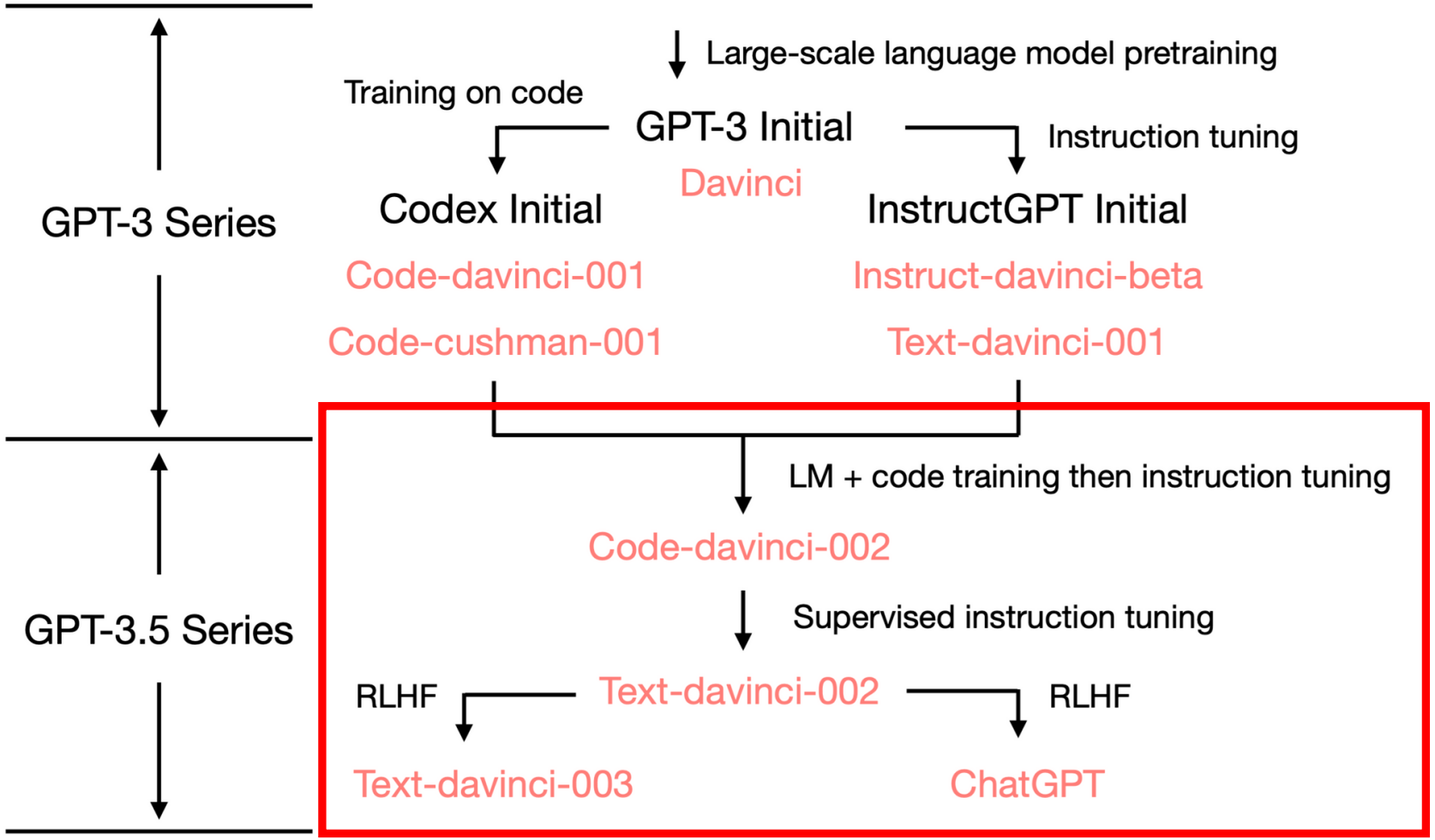
Public documentation: https://aka.ms/oai/models

Start with gpt-35-turbo.

| External name | Model names |
|---|---|
| GPT 3 | text-ada-001, text-babbage-001, text-curie-001, text-davinci-001 |
| GPT 3.5 | code-davinci-002, text-davinci-002, text-davinci-003 |
| Codex | code-cushman-001, code-cushman-002, code-davinci-002 |
| ChatGPT | gpt-35-turbo |
| GPT 4 | gpt-4, gpt-4-32k |

MODELS OVERVIEW

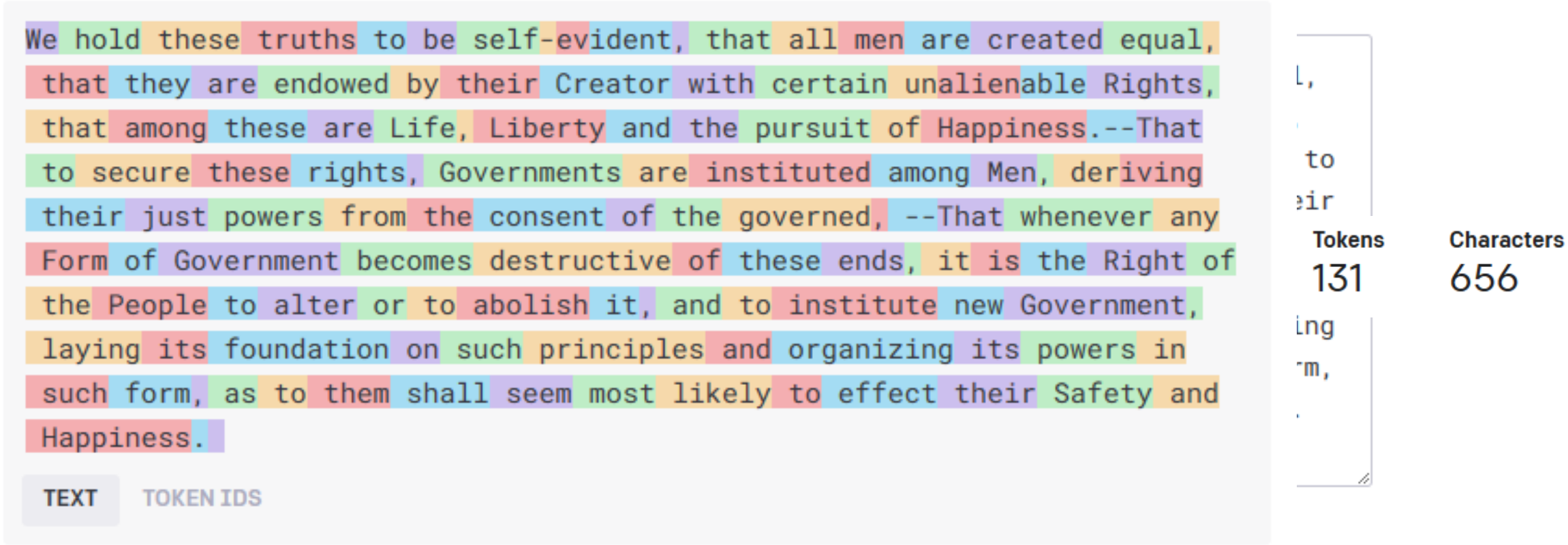
# GPT-3 FAMILY TREE



MODELS OVERVIEW

# TOKENS

- Inputs and Outputs are numeric vectors
- A tokenizer converts between strings and IDs



# INSTRUCTIONS VS COMPLETIONS

GPT models tend to display disparate behavior

- Artifact of training data
- RLHF → Instructions
- Next Token → Completions

Use the right “mode” for your scenario

MODELS OVERVIEW

# BASIC PROMPT COMPONENTS

- Instructions
- Content (translate, summarize)
- Cue (help completion)
- Examples (Few-Shot “Learning”)
- Structured Data

# CHALLENGES

**Hallucination**

- “Falsehoods” generated by the model
- Creativity’s double-edged sword
- Solutions:
    - Parameters
    - Prompt techniques
    - RLHF

**Bias + Toxicity**

- Artifact of unsupervised training data
- Solutions:
    - Data filtering
    - Post process
    - Prompting techniques

# HALLUCINATION (BUT DISCUSS “GROUNDING”)

The creativity of the model can work against it by generating reasonable responses that aren’t fact-based or grounded in the source data.

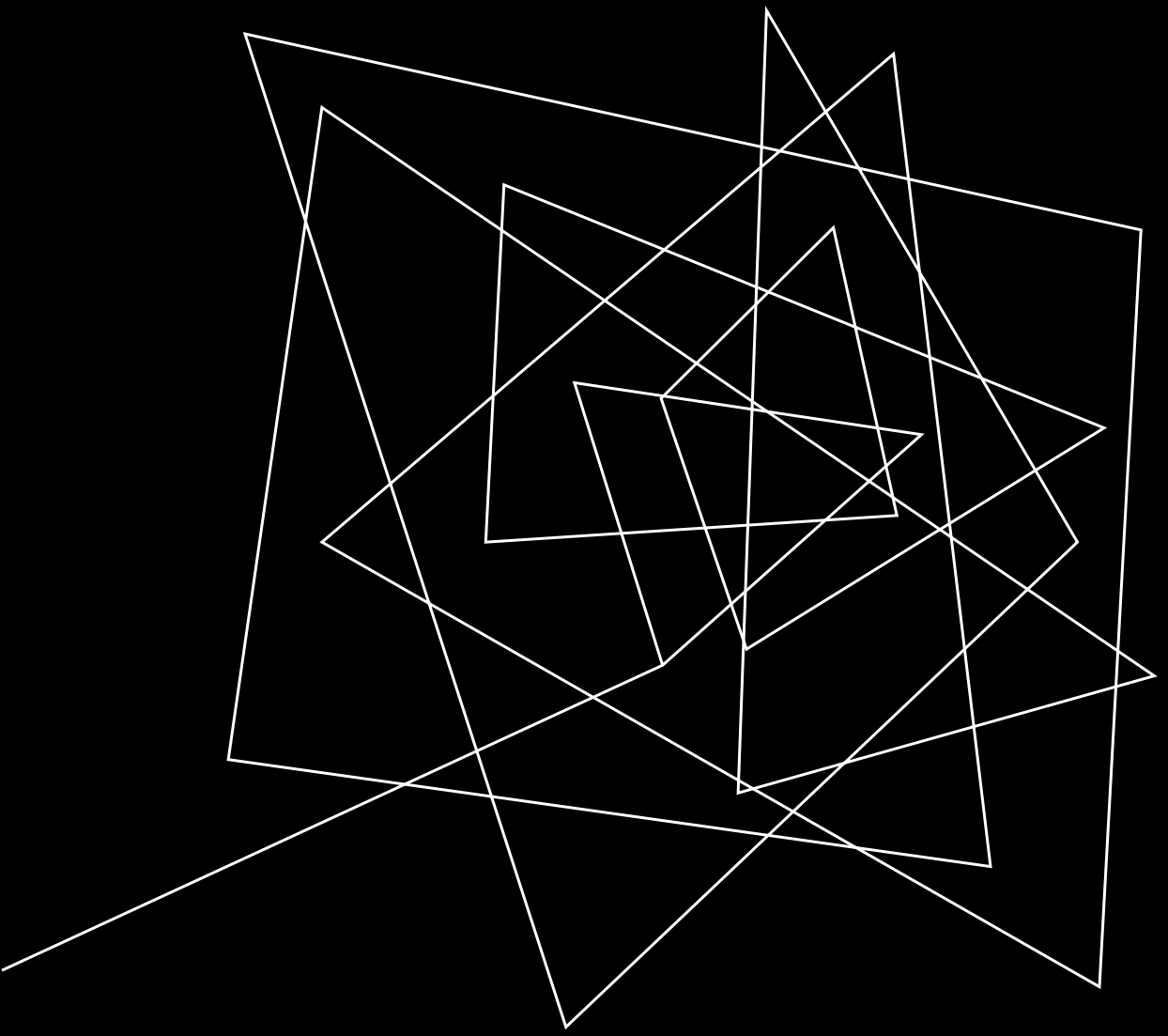
Workarounds to reduce hallucination/improve grounding:

- Use Bing “More Precise” mode (or the model DV3-PPO 32k aka Sydney PPO)
- Expand the grounding data
- Ask for inline citations. Asking the model to repeat some of its inputs, such as the grounding information, before attempting to generate an output.
- Reduce temperature
- Reduce max length
- Use affordances/skills/plugins for things like math
- Frame the problem more as a summarization task than a question-answering task
- Consider a rewrite step.
- Chain-of-thought prompting

# Hallucination

Bing Chat in “More Creative” mode:
“Write a three-paragraph bio for <your name>, including information about work accomplishments, awards received, and personal life.”

DEMO #1



# COMPLETIONS

COMPLETIONS

# GUIDELINES FOR CREATING ROBUST PROMPTS

There are three basic guidelines for creating useful prompts:

- **Show and tell.** Make it clear what you want either through instructions, examples, or a combination of the two. If you want the model to rank a list of items in alphabetical order or to classify a paragraph by sentiment, include these details in your prompt to show the model.
- **Provide quality data**. If you're trying to build a classifier or get the model to follow a pattern, make sure there are enough examples. Be sure to proofread your examples. The model is smart enough to resolve basic spelling mistakes and give you a meaningful response. Conversely, the model might assume the mistakes are intentional, which can affect the response.
- **Check your settings**. Probability settings, such as Temperature and Top P, control how deterministic the model is in generating a response. If you're asking for a response where there's only one right answer, you should specify lower values for these settings. If you're looking for a response that's not obvious, you might want to use higher values. The most common mistake users make with these settings is assuming they control "cleverness" or "creativity" in the model response.

COMPLETIONS

# CLASSIFY TEXT

To create a text classifier with the API, you provide a description of the task and provide a few examples. In this demonstration, you show the API how to classify the sentiment of text messages. The sentiment expresses the overall feeling or expression in the text.

```
This is a text message sentiment classifier

Message: "I loved the new adventure movie!"
Sentiment: Positive

Message: "I hate it when my phone battery dies."
Sentiment: Negative

Message: "My day has been 👍"
Sentiment: Positive

Message: "This is the link to the article"
Sentiment: Neutral

Message: "This new music video is unreal"
Sentiment:
```

COMPLETIONS

# GUIDELINES FOR DESIGNING TEXT CLASSIFIERS

This demonstration reveals several guidelines for designing classifiers:

- **Use plain language to describe your inputs and outputs.** Use plain language for the input "Message" and the expected value that expresses the "Sentiment." For best practices, start with plain language descriptions. You can often use shorthand or keys to indicate the input and output when building your prompt, but it's best to start by being as descriptive as possible. Then you can work backwards and remove extra words as long as the performance to the prompt is consistent.
- **Show the API how to respond to any case.** The demonstration provides multiple outcomes: "Positive," "Negative," and "Neutral." Supporting a neutral outcome is important because there are many cases where even a human can have difficulty determining if something is positive or negative.
- **Use emoji and text, per the common expression.** The demonstration shows that the classifier can be a mix of text and emoji 👍. The API reads emoji and can even convert expressions to and from them. For the best response, use common forms of expression for your examples.
- **Use fewer examples for familiar tasks.** This classifier provides only a handful of examples because the API already has an understanding of sentiment and the concept of a text message. If you're building a classifier for something the API might not be familiar with, it might be necessary to provide more examples.

COMPLETIONS

# MULTIPLE RESULTS FROM A SINGLE API CALL

Now that you understand how to build a classifier, let's expand on the first demonstration to make it more efficient. You want to be able to use the classifier to get multiple results back from a single API call.

This is a text message sentiment classifier

Message: "I loved the new adventure movie!"
Sentiment: Positive

Message: "I hate it when my phone battery dies"
Sentiment: Negative

Message: "My day has been 👍"
Sentiment: Positive

Message: "This is the link to the article"
Sentiment: Neutral

Message text
1. "I loved the new adventure movie!"
2. "I hate it when my phone battery dies"
3. "My day has been 👍"
4. "This is the link to the article"
5. "This new music video is unreal"

Message sentiment ratings:
1: Positive
2: Negative
3: Positive
4: Neutral
5: Positive

Message text
1. "He doesn't like homework"
2. "The taxi is late. She's angry 😡"
3. "I can't wait for the weekend!!!"
4. "My cat is adorable 💙 💙"
5. "Let's try chocolate bananas"

Message sentiment ratings:
1.

COMPLETIONS

# TRIGGER IDEAS

One of the most powerful yet simplest tasks you can accomplish with the API is generating new ideas or versions of input. Suppose you're writing a mystery novel and you need some story ideas. You can give the API a list of a few ideas and it tries to add more ideas to your list. The API can create business plans, character descriptions, marketing slogans, and much more from just a small handful of examples.

Ideas involving education and virtual reality

1. Virtual Mars
Students get to explore Mars via virtual reality and go on missions to collect and catalog what they see.

2.

COMPLETIONS

# GUIDELINES FOR TRIGGERING IDEAS

Although this demonstration uses a simple prompt, it highlights several guidelines for triggering new ideas:

- **Explain the intent of the list.** Similar to the demonstration for the text classifier, you start by telling the API what the list is about. This approach helps the API to focus on completing the list rather than trying to determine patterns by analyzing the text.
- **Set the pattern for the items in the list.** When you provide a one-sentence description, the API tries to follow that pattern when generating new items for the list. If you want a more verbose response, you need to establish that intent with more detailed text input to the API.
- **Prompt the API with an incomplete entry to trigger new ideas.** When the API encounters text that seems incomplete, such as the prompt text "2.," it first tries to determine any text that might complete the entry. Because the demonstration had a list title and an example with the number "1." and accompanying text, the API interpreted the incomplete prompt text "2." as a request to continue adding items to the list.
- **Explore advanced generation techniques.** You can improve the quality of the responses by making a longer more diverse list in your prompt. One approach is to start with one example, let the API generate more examples, and then select the examples you like best and add them to the list. A few more high-quality variations in your examples can dramatically improve the quality of the responses.

COMPLETIONS

# CONDUCT CONVERSATIONS

Starting with the release of GPT-35-Turbo and GPT-4, we recommend that you create conversational generation and chatbots by using models that support the chat completion endpoint. The chat completion models and endpoint require a different input structure than the completion endpoint.

The API is adept at carrying on conversations with humans and even with itself. With just a few lines of instruction, the API can perform as a customer service chatbot that intelligently answers questions without getting flustered, or a wise-cracking conversation partner that makes jokes and puns. The key is to tell the API how it should behave and then provide a few examples.

```
The following is a conversation with an AI assistant. The assistant is helpful, creative, clever, and very friendly.

Human: Hello, who are you?
AI: I am an AI created by OpenAI. How can I help you today?
Human:
```

COMPLETIONS

# CONDUCT CONVERSATIONS

Let's look at a variation for a chatbot named "Cramer," an amusing and somewhat helpful virtual assistant. To help the API understand the character of the role, you provide a few examples of questions and answers. All it takes is just a few sarcastic responses and the API can pick up the pattern and provide an endless number of similar responses.

```
Cramer is a chatbot that reluctantly answers questions.

###
User: How many pounds are in a kilogram?
Cramer: This again? There are 2.2 pounds in a kilogram. Please make a note of this.
###
User: What does HTML stand for?
Cramer: Was Google too busy? Hypertext Markup Language. The T is for try to ask better questions in the future.
###
User: When did the first airplane fly?
Cramer: On December 17, 1903, Wilbur and Orville Wright made the first flights. I wish they'd come and take me away.
###
User: Who was the first man in space?
Cramer:
```

COMPLETIONS

# GUIDELINES FOR DESIGNING CONVERSATIONS

Our demonstrations show how easily you can create a chatbot that's capable of carrying on a conversation. Although it looks simple, this approach follows several important guidelines:

- **Define the intent of the conversation.** Just like the other prompts, you describe the intent of the interaction to the API. In this case, "a conversation." This input prepares the API to process subsequent input according to the initial intent.
- **Tell the API how to behave.** A key detail in this demonstration is the explicit instructions for how the API should interact: "The assistant is helpful, creative, clever, and very friendly." Without your explicit instructions, the API might stray and mimic the human it's interacting with. The API might become unfriendly or exhibit other undesirable behaviour.
- **Give the API an identity.** At the start, you have the API respond as an AI created by OpenAI. While the API has no intrinsic identity, the character description helps the API respond in a way that's as close to the truth as possible. You can use character identity descriptions in other ways to create different kinds of chatbots. If you tell the API to respond as a research scientist in biology, you receive intelligent and thoughtful comments from the API similar to what you'd expect from someone with that background.

COMPLETIONS

# TRANSFORM TEXT

The API is a language model that's familiar with various ways that words and character identities can be used to express information. The knowledge data supports transforming text from natural language into code, and translating between other languages and English. The API is also able to understand content on a level that allows it to summarize, convert, and express it in different ways. Let's look at a few examples.

COMPLETIONS

# TRANSFORM TEXT - TRANSLATE FROM ONE LANGUAGE TO ANOTHER

This example works because the API already has a grasp of the French language. You don't need to try to teach the language to the API. You just need to provide enough examples to help the API understand your request to convert from one language to another.

If you want to translate from English to a language the API doesn't recognize, you need to provide the API with more examples and a fine-tuned model that can produce fluent translations.

```
English: I do not speak French.
French: Je ne parle pas français.
English: See you later!
French: À tout à l'heure!
English: Where is a good restaurant?
French: Où est un bon restaurant?
English: What rooms do you have available?
French: Quelles chambres avez-vous de disponible?
English:
```

COMPLETIONS

# TRANSFORM TEXT - CONVERT BETWEEN TEXT AND EMOJI

This demonstration converts the name of a movie from text into emoji characters. This example shows the adaptability of the API to pick up patterns and work with other characters.

Carpool Time: 👨 👴 👩 🚙 🕒
Robots in Cars: 🚙 🤖
Super Femme: 🦸 🦸 🦸 🦸 🦸
Webs of the Spider: 🕸 🕷 🕸 🕸 🕷 🕸
The Three Bears: 🐻 🐼 🐻
Mobster Family: 👨 👩 👧 🕵 💂 💥
Arrows and Swords: 🏹 🗡 🗡 🏹
Snowmobiles:

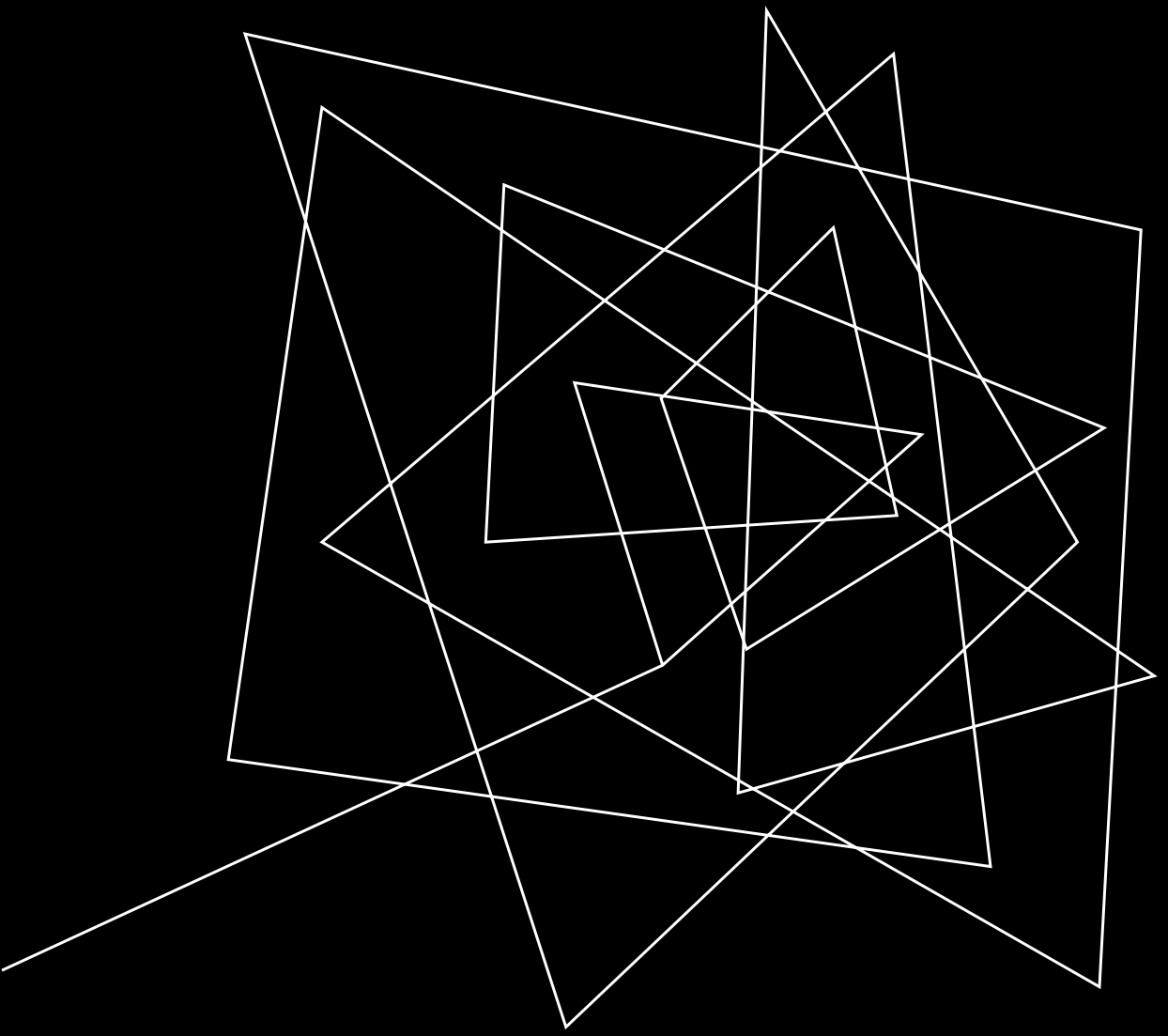
COMPLETIONS

# TRANSFORM TEXT – SUMMARIZE TEXT

The API can grasp the context of text and rephrase it in different ways. In this demonstration, the API takes a block of text and creates an explanation that's understandable by a primary-age child. This example illustrates that the API has a deep grasp of language.

```
My ten-year-old asked me what this passage means:
"""
A neutron star is the collapsed core of a massive supergiant star, which had a total mass of between 10 and 25 solar masses, possibly more if the star was especially metal-rich.[1] Neutron stars are the smallest and densest stellar objects, excluding black holes and hypothetical white holes, quark stars, and strange stars.[2] Neutron stars have a radius on the order of 10 kilometres (6.2 mi) and a mass of about 1.4 solar masses.[3] They result from the supernova explosion of a massive star, combined with gravitational collapse, that compresses the core past white dwarf star density to that of atomic nuclei.
"""

I rephrased it for him, in plain language a ten-year-old can understand:
"""
```



# TOOLS

TOOLS

# LANGCHAIN

Open Source **framework** for developing applications powered by language models that are **data-aware** (can connect to data sources) and **agentic** (can interact with the environment).

**Main Concepts**

- **Modules** (modular abstractions) – components that handle common tasks such as prompts, indexes and memory.
- **Chains** – assembly of components put together to accomplish a use case.
- **Agents** – special type of chain that can decide which tools to use depending on the user input.
- **Tools** – functions that agents can use to interact with the world. Can be generic utilities, chains or other agents.

**Features**

- Huge ecosystem and community
- Support for multiple LLMs, document loaders, text splitters, retrievers and vector stores
- Offers several use cases ready to implement
- Support for Python and JavaScript

TOOLS

# SEMANTIC KERNEL

Open Source **SDK**, lead by Microsoft, to integrate LLMs with conventional programming languages infusing applications with complex skills like prompt chaining, recursive reasoning, memory, semantic indexing and more.

**Main Concepts**

- **Kernel** – orchestrator of a user's Ask (goal). Fulfills the goal using available skills, memories and connectors.
- **Planner** – component that breaks down the goal into steps based upon available resources.
- **Skill** – domain of expertise made available to the kernel as a function or group of functions.
- **Memory** – provides context for the Ask. Can be accessed as key-value pairs, local storage or semantic memory search.

**Features**

- Highly extensible
- Support for multiple models, skills and connectors
- Support for C#, Python and JavaScript

TOOLS

# PROMPT FLOW

Prompt engineering tool included in Azure Machine Learning Studio.

**Main Features**

- Create AI workflows that connect to various language models and data sources.
- One platform to build, tune, evaluate, deploy, and test AI workflows.
- Visual representation of prompt workflows using graphs. Each step is mapped to a notebook-like code cell.
- Evaluate the quality of AI workflows with rich set of pre-built metrics.
- Easy prompt tuning, comparison, tracking.