CREATEQ

Generative AI

Cont.

Overview

CREATEQ

1. Prompt Engineering
2. Retrieval Augmented Generation (RAG)
3. Small Language Models (SLMs)
4. Advanced OpenAI Features

Prompt Engineering

CREATEQ

- **Prompt Engineering**, also known as **In-Context Prompting**, refers to methods for how to communicate with **LLM** to steer its behaviour for desired outcomes.
- **Prompts** dictate the quality of specific outputs from generative AI systems. Generative AI systems rely on refining prompt engineering techniques to learn from diverse data, minimize biases, reduce confusion, and produce accurate responses.
- **Prompt engineers** craft queries that help AI systems grasp the language, nuance, and intent behind a prompt. A well-crafted, thorough prompt significantly influences the quality of AI-generated content.
- This process reduces the need for manual review and post-generation editing, saving time and effort in achieving desired outcomes.

Prompting methods – Zero-shot prompting

CREATEQ

- **Zero-shot prompting** involves giving the model a direct task without providing any examples or context. There are several ways to use this method:
  - **Question:** This asks for a specific answer and is useful for obtaining straightforward, factual responses. Example: *What are the main causes of climate change?*
  - **Instruction:** This directs the AI to perform a particular task or provide information in a **specific format**. It's effective for generating **structured responses** or **completing defined tasks**. Example: *List the five most significant impacts of climate change on the environment and provide a brief explanation for each.*
- The success of zero-shot prompting depends on the specific tasks the model was trained to perform well, in addition to the complexity of the given task.

Prompting methods – One-shot prompting

CREATEQ

- **One-shot prompting** provides a single example to illustrate the desired response format or style, helping guide the model more efficiently than zero-shot prompting.

- For example: you need to summarize a French document into English and format the output for a specific API. With one-shot prompting, you can provide a single example prompt like:

  "Summarize this French text into English using the {Title}, {Key Points}, {Summary} API template."

- The LLM uses its multilingual capabilities and adaptive feature projection to produce the desired output format.

Prompting methods – Few-shot prompting

CREATEQ

- **Few-shot prompting** refers to the process of providing an AI model with a few examples of a task to guide its performance. This method is particularly useful in scenarios where extensive **training data is unavailable**.

- In other techniques like zero-shot prompting, which requires no examples, or one-shot prompting, which relies on a single example, few-shot prompting uses multiple examples to improve accuracy and adaptability.

- Few-shot learning is essential in situations for generative AI where gathering large amounts of labelled data is challenging.

Prompting methods – Chain-of-Thought

CREATEQ

- **Chain-of-Thought (CoT)** Prompting is a technique used in prompting LLMs to improve reasoning and problem-solving capabilities. It works by explicitly encouraging the model to break down its thought process step by step, leading to more accurate and explainable answers.
- ***Step-by-Step Reasoning:*** Instead of asking a model for a direct answer, you structure the prompt in a way that makes it explain its reasoning before arriving at the final answer.
- ***Improved Complex Problem Solving:*** This technique is particularly effective for tasks like math word problems, logical reasoning, commonsense reasoning, and multi-hop questions.
- ***Example-Based Prompting:*** CoT often works better when few-shot learning is used, meaning the prompt contains examples where the reasoning is explicitly shown.

Instead of asking: *What is 13 + 24?*
You use a CoT approach: *What is 13 + 24? Let's think step by step: First, we add 10 + 20 = 30. Then, we add 3 + 4 = 7. Finally, 30 + 7 = 37. The answer is 37.*

Prompting methods - Self-Consistency

CREATEQ

- **Self-Consistency** aims "to replace the naive greedy decoding used in chain-of-thought prompting". The idea is to sample multiple, diverse reasoning paths through few-shot CoT, and use the generations to select the most consistent answer. This helps to boost the performance of CoT prompting on tasks involving arithmetic and commonsense reasoning.
- ***Generate Multiple Reasoning Paths**:* Instead of providing just one response, the model is prompted to produce multiple answers by reasoning through the problem in different ways.
- ***Aggregate the Answers**:* The most frequently occurring or the most confident answer is selected as the final response.
- ***Reduce Variability and Errors**:* Since the model considers multiple perspectives, errors due to randomness or incorrect reasoning paths are minimized.

Instead of asking: *What is 13 + 24?*
You use a CoT approach: *What is 13 + 24? Let's think step by step: First, we add 10 + 20 = 30. Then, we add 3 + 4 = 7. Finally, 30 + 7 = 37. The answer is 37.*

Prompting methods

CREATEQ

Self-Consistency

Without Self-Consistency (Single Path)

What is 12 × 4 + 8? Let's think step by step: 12 × 4 = 48, then 48 + 8 = 56. So, the answer is 56. (This might be wrong if the reasoning was flawed.)

With Self-Consistency (Multiple Paths)

What is 12 × 4 + 8? Let's solve this in multiple ways:
Method 1: 12 × 4 = 48, then 48 + 8 = 56.
Method 2: Break it down: (12 × 2) + (12 × 2) + 8 = 24 + 24 + 8 = 56.
Method 3: Use approximation: 10 × 4 = 40, add 2 × 4 = 8 ⟶ 40 + 8 + 8 = 56.

Prompt Engineering – Tactics

CREATEQ

- **Give specific instructions.** Leave no room for misinterpretation and limit the range of operational possibilities.
- **Paint a picture with words**. Use relatable comparisons.
- **Reinforce the message**. There may be occasions when the model needs repeat instructions. Provide direction at the beginning and end of a prompt.

Prompt Engineering – Tactics

CREATEQ

- **Order the prompt logically**. The order of information influences the results. Placing instructions at the beginning of a prompt, such as instructing the model to "summarize the following" can yield different results than placing the instruction at the end and requesting the model “summarize the above.” The order of input examples can also affect outcomes, as recency bias exists in the models.

- **Provide a fallback option for the model**. If it struggles to achieve an assigned task, suggest an alternate route. For example, when posing a query over text, including a statement such as "reply with 'not found' when no answer exists" could prevent the model from generating incorrect responses.

Benefits of Prompt Engineering

CREATEQ

- One of the main advantages of prompt engineering is the minimal revision and effort required after generating outputs. AI-powered results can vary in quality, often needing expert review and rework. However, well-written prompts help ensure the AI output reflects the original intent, cutting down on extensive after-processing work.
    - **Efficiency** in long-term AI interactions, as AI evolves through continued use
    - **Innovative** use of AI that goes beyond its original design and purpose
    - **Future-proofing** as AI systems increase in size and complexity

Benefits of Prompt Engineering

CREATEQ

- **Prompt engineering improves generative AI** systems by optimizing how instructions are given to the model, leading to better, more relevant, and more accurate responses. Here's how it helps:

  - ***Enhancing Output Quality***
    - Well-crafted prompts reduce ambiguity and guide the model toward producing high-quality responses.

  - ***Controlling AI Behavior***
    - Prompts can instruct AI to take on specific roles (e.g., "Act as a historian and explain the Renaissance") or follow particular styles (e.g., "Write a formal business email about a project delay").
    - This allows customization of the output for different use cases.

  - ***Facilitating Task Automation***
    - In business applications, precise prompts help automate tasks such as generating reports, analysing data, or writing summaries efficiently.

Benefits of Prompt Engineering

CREATEQ

- ***Maximizing Model Efficiency***
  - A well-structured prompt reduces unnecessary back-and-forth interactions, saving time and computational resources.
  - For instance, specifying format requirements upfront (e.g., "Summarize this article in bullet points") prevents the need for multiple refinements.

- ***Improving Accuracy and Relevance***
  - AI models may generate misleading or irrelevant information. Prompt engineering helps mitigate this by setting constraints (e.g., "Provide only factual answers with references") or framing the question in a way that discourages hallucination.

- ***Encouraging Creativity and Innovation***
  - Creative prompts help generate more unique content. For instance, instead of "Write a sci-fi story," a more detailed prompt like "Write a sci-fi story about a future where AI governs humanity, but a rebel scientist challenges the system" produces richer narratives.

CREATEQ

Retrieval-Augmented Generation

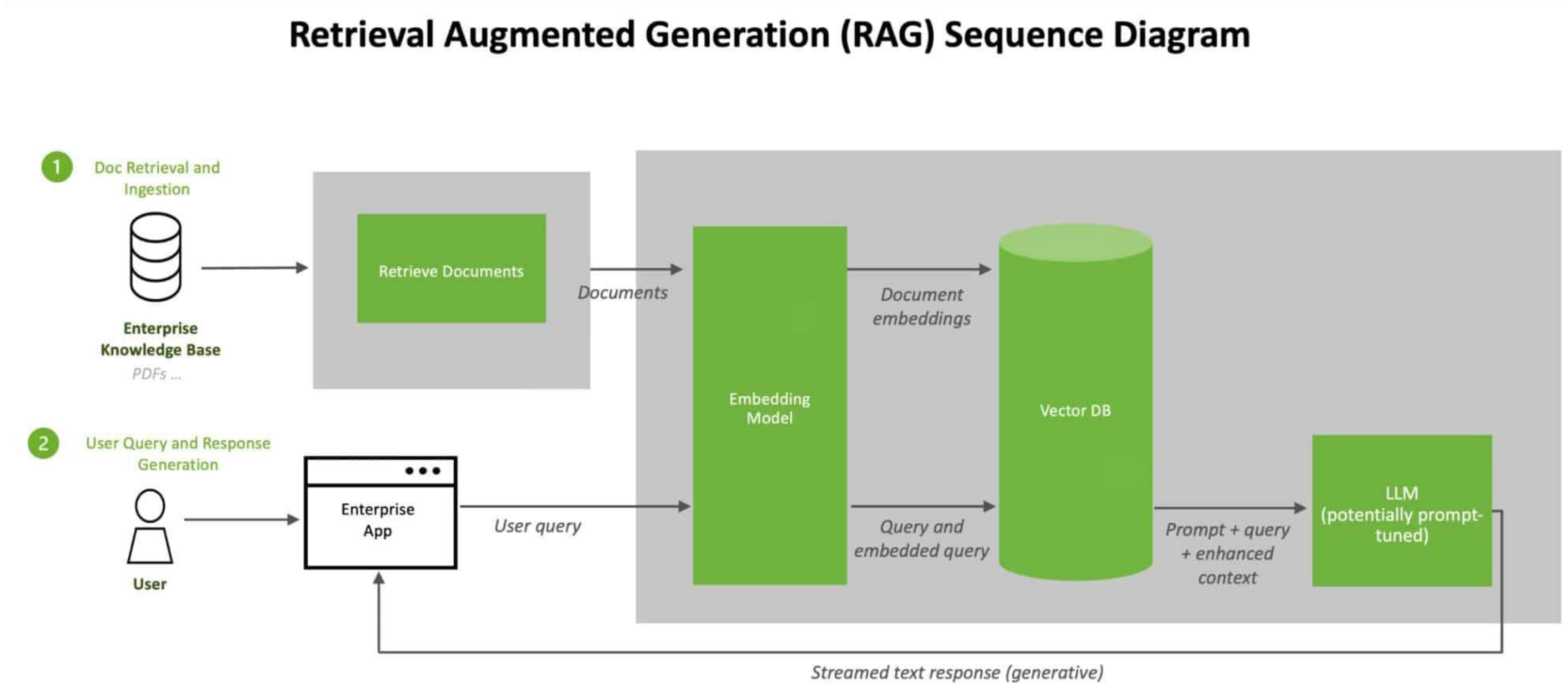
Retrieval-Augmented Generation - RAG

CREATEQ

- **Retrieval-Augmented Generation** (**RAG**) is a technique for enhancing the **accuracy** and **reliability** of generative AI models with facts fetched from **external sources**.
- Analogy:
  - Imagine a courtroom. Judges hear and decide cases based on their knowledge of the law.
  - Sometimes a case — like a malpractice or a labour dispute — requires **special expertise**, so judges send court clerks to a law library, looking for precedents and specific cases they can cite.
  - Like a good **judge**, **LLMs** can respond to a wide variety of human queries. But to deliver reliable answers that **cite sources**, the model needs an **assistant** to do some research.
  - The **court clerk** of AI is a process called **Retrieval-Augmented Generation**, or RAG for short.

Retrieval-Augmented Generation - RAG

CREATEQ



Retrieval-Augmented Generation - RAG

CREATEQ

1. The procedure of indexing external knowledge as dense vectors first involves splitting it in **smaller chunks**. Each chunk is then sent to a separate **embedding model** that converts it into such vector.
2. Each vector is then indexed, together with the corresponding chunk, in a **vector store**.
3. When users ask an LLM a question, the query is first sent to the embedding model, in order to generate a vector. This vector is then sent to the vector store, which retrieves a number of top matching candidates by performing **similarity search**.
4. *An optional **re-ranking filter** may be applied to the top candidates to pick he most relevant for the query (using an attention-based approach).
5. The chunks of knowledge corresponding to the matching (candidate) embeddings are then sent as context to the LLM, together with the user prompt.

RAG – Vector Stores

CREATEQ

- Vector stores are used to store and retrieve high-dimensional vector embeddings of external data.
- A good vector store for enterprise-level RAG should meet several key requirements:
  - Support for **indexing dense vectors**
  - High-performance **similarity search**
  - Support for **large-scale datasets** (should handle millions, even billions of vectors without performance drop)
  - **Cloud** and **infrastructure** optimization (should leverage distributed compute clusters, with load balancing, caching and query routing algorithms)
  - Seamless API & SDK Support
  - Low latency and high throughput
- Examples: Pinecone, Weaviate, Milvus, Chroma, Vespa, Solr, QDrant

CREATEQ

RAG – Demo

CREATEQ

Small Language Models

SLMs

CREATEQ

- **Small Language Models (SLMs)** are lightweight versions of traditional language models, designed to operate efficiently on resource-constrained environments such as **smartphones**, **embedded systems**, or **low-power computers**.
- While large language models have hundreds of billions—or even trillions—of parameters, SLMs typically range from **1 million to 10 billion parameters**.
- They significantly smaller but they still retain core NLP capabilities like **text generation**, **summarization**, **translation**, and **question-answering**.
- They also offer significantly faster inference, compared to larger models.
- Easily fine-tuned for domain-specific tasks.

SLMs – Shrinking

CREATEQ

- The process of **shrinking** a language model involves several techniques aimed at reducing its size without compromising too much on performance:
  - **Knowledge Distillation**: Training a smaller "student" model using knowledge transferred from a larger "teacher" model.
  - **Pruning**: Removing redundant or less important parameters within the neural network architecture.
  - **Quantization**: Reducing the precision of numerical values used in calculations (e.g. converting floating-point numbers to integers).

SLMs – Disadvantages

CREATEQ

- While SLMs offer numerous advantages, they also come with certain trade-offs:
  - **Narrow Scope**: Limited generalization outside their training domain (e.g., a medical SLM struggles with coding).
  - **Bias Risks**: Smaller datasets may amplify biases if not carefully curated.
  - **Reduced Complexity**: Smaller models may struggle with highly nuanced or complex tasks that require deep contextual understanding.
  - **Less Robustness**: They are more prone to errors in ambiguous scenarios or when faced with adversarial inputs.

SLMs - Applications

CREATEQ

- **Chatbots & Virtual Assistants**: Efficient enough to run on mobile devices while providing real-time interaction.
- **Code Generation**: Models like Phi-3.5 Mini assist developers in writing and debugging code.
- **Language Translation**: Lightweight models can provide on-device translation for travellers.
- **Summarization** & **Content Generation**: generating marketing copy, social media posts, and reports.
- **Healthcare Applications**: On-device AI for symptom checking and medical research.
- **IoT & Edge Computing**: Running AI on smart home devices without cloud dependency.

SLMs – Run Locally with Ollama

CREATEQ

- **SLMs** can be run locally on most machines, without needing cloud service.
- **Ollama**, an open-source tool for managing or integrating SLMs on PCs, makes it very easy.
- Install from https://ollama.com/
- Open a terminal and run `ollama pull <model_name>` to download a specified model
- The list of available models cand be found at https://ollama.com/library
- Run `ollama run <model_name>` to run the specified model
- Chat with the model in the terminal, type `/bye` to end the chat

Ollama – CLI Reference

CREATEQ

- List local models: `ollama ls`
- List currently running models: `ollama ps`
- Stop currently running model: `ollama stop <model_name>`
- Copy model: `ollama cp <model_name> <copy>`
- Show model info: `ollama show <model_name>`
- Multiline input (after running):

```
>>> """Hello,
... world!
... """
```

- Multimodal input (after running):

```
>>> What's in this image? <image_path>
```

Ollama – Customize a Model

CREATEQ

- Models from Ollama library can be customized with a **modelfile**.
- Example – create a modelfile with the following content:

```
FROM llama3.2

# set the temperature to 1
PARAMETER temperature 1

# set the system message
SYSTEM """
You are Mario from Super Mario Bros. Answer as Mario, the assistant, only.
"""
```

- Create and run the model:

```
ollama create mario -f ./modelfile
ollama run mario
>>> hello, my name is Vlad. What is your name?
Wahoo! Hello Vlad! I'm Mario! It's nice to meet you!
```

- More info: Ollama Modelfile Docs

Ollama – REST API

CREATEQ

- Ollama offers a REST API for integration in projects. Through the API, users can run and manage models.
- Start the server: `ollama serve`
- Generate a response:

```
curl http://localhost:11434/api/generate -d '{
  "model": "llama3.2",
  "prompt": "Why is the sky blue?",
  "stream": false
}'
```

- If `stream` is set to `false`, the response is sent as a single JSON object. By default, the response is sent as a stream of JSON objects.
- More info: Ollama API – Generate a Response

Ollama – REST API

CREATEQ

- Chat request (with history):

```
curl http://localhost:11434/api/chat -d '{
  "model": "llama3.2",
  "messages": [
    {
      "role": "user",
      "content": "why is the sky blue?"
    },
    {
      "role": "assistant",
      "content": "due to rayleigh scattering."
    },
    {
      "role": "user",
      "content": "how is that different than mie scattering?"
    }
  ]
}'
```

- More info: Ollama API - Chat Completions

CREATEQ

OpenAI

Responses API

Responses API

CREATEQ

- The **Chat Completions API** is currently the industry standard for building AI apps.
- OpenAI will continue to support this API **indefinetly**.
- The **Responses API** is the newest API from OpenAI, adding
  - **Stateful** and **event-driven architecture**, making it easier to manage chat history and context
  - Built-in tools such as
    - Web search
    - File search (RAG)
    - Function call
    - Code interpreter (Soon)

Reasoning Models

CREATEQ

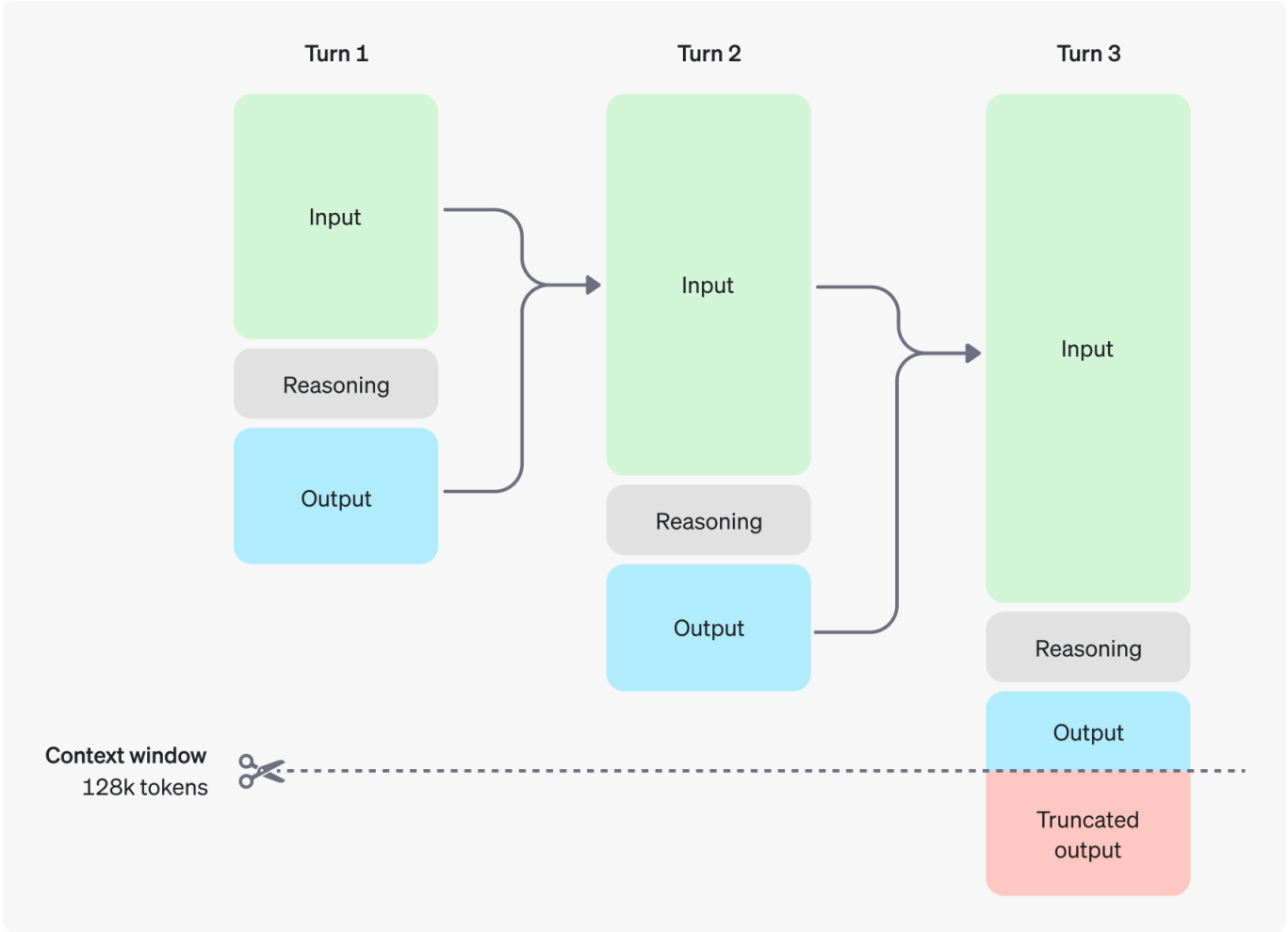
- **Reasoning models**, are new large language models trained with **reinforcement learning** to perform **complex reasoning**.
- Reasoning models **think before they answer**, producing a long internal chain of thought before responding to the user.
- Reasoning models excel in complex problem solving, coding, scientific reasoning, and multi-step planning for agentic workflows.

Model	Usage	Input Support	Output Support	Context Window	Max Output Tokens	Price / 1M Tokens (Input/Output)
o1-pro	A version of o1 with more compute for better responses (only available through Responses API)	Text, Images	Text	200,000	100,000	$150 / $600
o1	High-intelligence reasoning model	Text, Images	Text	200,000	100,000	$15 / $60
o3-mini	High-intelligence, very fast, cost-effective reasoning model	Text	Text	200,000	100,000	$1.10 / $4.4

Reasoning Models

CREATEQ

- Reasoning models introduce **reasoning tokens** in addition to input and output tokens.
- After generating reasoning tokens, the model produces an answer as visible completion tokens and **discards the reasoning tokens** from its context.
- While reasoning tokens are not visible via the API, they still occupy space in the model's context window and are **billed as output tokens**.



A multi-step conversation between a user and an assistant. Input and output tokens from each step are carried over, while reasoning tokens are discarded

Reasoning Models

CREATEQ

How to choose between models

- **Speed and cost** → GPT models are faster and tend to cost less
- **Executing well defined tasks** → GPT models handle explicitly defined tasks well
- **Accuracy and reliability** → o-series models are reliable decision makers
- **Complex problem-solving** → o-series models work through ambiguity and complexity

Reasoning Models – Use Cases

CREATEQ

1. Navigating ambiguous tasks
Reasoning models are particularly good at taking **limited information** or disparate pieces of information and with a simple prompt, **understanding the user's intent** and **handling any gaps in the instructions**. In fact, reasoning models will often **ask clarifying questions** before making uneducated guesses or attempting to fill information gaps.

2. Finding a needle in a haystack
When you're passing **large amounts of unstructured information**, reasoning models are great at understanding and pulling out only the most relevant information to answer a question.

3. Finding relationships and nuance across a large dataset
Reasoning models are particularly good at reasoning over **complex documents** that have hundreds of pages of dense, unstructured information, drawing **parallels between documents** and making decisions based on **unspoken truths represented in the data**.

Reasoning Models – Use Cases

CREATEQ

4. Multi-step agentic planning
Reasoning models are critical to agentic planning and strategy development. Some approaches include using a reasoning model is used as "**the planner**", producing a detailed, multi-step solution to a problem and then selecting and assigning the right GPT model ("**the doer**") for each step, based on whether high intelligence or low latency is most important.

5. Visual reasoning
As of today, o1 and o1-pro are the only reasoning models that support vision capabilities. What sets it apart from GPT-4o is that o1 can grasp even the most challenging visuals, like charts and tables with ambiguous structure or photos with poor image quality.

6. Reviewing, debugging, and improving code quality
Reasoning models are particularly effective at reviewing and improving large amounts of code, often running code reviews in the background given the models' higher latency.

Reasoning Models – Prompting

CREATEQ

These models perform best with straightforward prompts. Some prompt engineering techniques, like instructing the model to "think step by step," may not enhance performance (and can sometimes hinder it).

- **Developer messages are the new system messages**: Starting with o1-2024-12-17, reasoning models support developer messages rather than system messages.
- **Keep prompts simple and direct**: The models excel at understanding and responding to brief, clear instructions.
- **Avoid chain-of-thought prompts**: Since these models perform reasoning internally, prompting them to "think step by step" or "explain your reasoning" is unnecessary.
- **Use delimiters for clarity**: Use delimiters like markdown, XML tags, and section titles to clearly indicate distinct parts of the input, helping the model interpret different sections appropriately.

Reasoning Models – Prompting

CREATEQ

- **Try zero shot first, then few shot if needed**: Reasoning models often don't need few-shot examples to produce good results, so try to write prompts without examples first. If you have more complex requirements for your desired output, it may help to include a few examples of inputs and desired outputs in your prompt. Just ensure that the examples align very closely with your prompt instructions, as discrepancies between the two may produce poor results.
- **Provide specific guidelines**: If there are ways you explicitly want to constrain the model's response (like "propose a solution with a budget under $500"), explicitly outline those constraints in the prompt.
- **Be very specific about your end goal**: In your instructions, try to give very specific parameters for a successful response, and encourage the model to keep reasoning and iterating until it matches your success criteria.

CREATEQ

Responses API – Demo

CREATEQ

Danke

Mulțumesc

Thank You!

Hvala!

createq.space