What is the difference between Predictive/Discriminative AI and Generative AI

Predictive/Discriminative AI and Generative AI are distinct paradigms within artificial intelligence, serving different purposes and employing unique approaches to data modeling and problem-solving. Below is a detailed comparison:

Key Differences

Feature	Predictive/Discriminative AI	Generative AI	
Purpose	Predicts outcomes or classifies data into categories.	Creates new data instances resembling the original dataset.	
Data Modeling	Models the conditional probability $ P(Y	X) $, focusing on decision boundaries between classes.	Models the joint probability $ P(X, Y) $, learning the entire data distribution to generate new samples.
Learning Approach	Supervised learning, using labeled data for classification tasks.	Often unsupervised or semi-supervised, leveraging unlabeled data for creative synthesis.	
Output	Accurate predictions or classifications (e.g., spam detection, image recognition).	Novel outputs like text, images, audio, or videos (e.g., ChatGPT, DALL-E).	
Training Complexity	Faster and simpler to train; requires less computational power.	Computationally intensive due to the need for modeling complex distributions.	
Applications	Fraud detection, medical diagnoses, sentiment analysis, facial recognition.	Content creation, drug discovery, AI-generated art, and synthetic data generation.	

Detailed Explanations

Predictive/Discriminative AI

- **Definition:** Discriminative models focus on distinguishing between predefined classes of data by learning decision boundaries. Predictive AI extends this by forecasting future events based on historical patterns [1] [2] [3].
- **Examples:** Logistic regression, Support Vector Machines (SVMs), and neural networks for classification tasks like spam filtering or credit scoring [4] [3].
- **Advantages:**
  - High accuracy in classification tasks.
  - Faster training due to simpler modeling.
  - Robust against noisy or outlier data [5].
- **Limitations:** Limited applicability in scenarios requiring creative output or understanding complex data distributions.

Generative AI

- **Definition:** Generative models learn the underlying distribution of input data to create new instances that resemble the original dataset [6] [7] [5].
- **Examples:** Generative Adversarial Networks (GANs), Variational Autoencoders (VAEs), and autoregressive models like GPT for generating text, images, or audio [8] [9].
- **Advantages:**
  - Enables creativity and innovation in applications like content generation and drug discovery.
  - Effective in unsupervised settings with unlabeled data [5].
- **Limitations:**
  - Computationally intensive due to complex modeling requirements.
  - May struggle with generating realistic outputs if poorly trained [5].

Predictive AI vs Generative AI

Predictive AI focuses on forecasting outcomes based on historical trends (e.g., stock price predictions) while generative AI creates entirely new content (e.g., generating synthetic images) [1] [2]. Predictive models are typically used in structured environments requiring accuracy and reliability, whereas generative models thrive in creative domains where innovation is key.

Complementary Roles

Both paradigms are essential in advancing AI capabilities:

- Predictive/Discriminative AI excels in precise decision-making tasks.
- Generative AI drives creativity and exploration by synthesizing novel outputs.

Together, these approaches form a symbiotic relationship that pushes the boundaries of artificial intelligence across diverse industries.

⁂

What is an LLM?

A **Large Language Model (LLM)** is a type of artificial intelligence system designed for **natural language processing (NLP)** tasks. It leverages deep learning techniques, particularly the **transformer architecture**, to understand, generate, and predict human-like text. LLMs are trained on massive datasets containing trillions of words from diverse sources like Wikipedia, books, and web pages. These models excel in tasks such as text generation, summarization, translation, question answering, and more[10] [11] [12].

LLMs typically have billions or even trillions of parameters, which are the variables learned during training that enable the model to capture complex relationships between words and concepts[10] [12]. They are foundational models that can be fine-tuned for specific applications or adapted to perform tasks with minimal additional training[10] [13].

How are LLMs Trained?

Training an LLM involves multiple stages and techniques aimed at building a robust model capable of understanding and generating human-like text. Below is a breakdown of the process:

1. Data Collection and Preprocessing

- **Corpus Selection:** The model is trained on vast amounts of text data (e.g., Wikipedia, Common Crawl) to ensure diversity and coverage[12] [14].
- **Tokenization:** Text is broken down into smaller units called tokens (e.g., words or subwords) using techniques like Byte Pair Encoding (BPE) or SentencePiece. This helps the model process sequences effectively[14] [15].
- **Standardization:** Data is cleaned and standardized to remove noise and inconsistencies[15].

2. Pre-Training

- **Objective:** Pre-training involves unsupervised learning where the model predicts missing words or the next word in a sequence. This helps it learn statistical patterns, grammar, syntax, and semantics[14] [16].
- **Transformer Architecture:** The model uses self-attention mechanisms within transformers to understand relationships between tokens in a sequence. This architecture allows parallel processing of sequences, making training efficient[12] [17].
- **Parameter Optimization:** During pre-training, weights and biases in the neural network are adjusted iteratively using optimization algorithms like Adam or SGD to minimize prediction errors[14] [17].

3. Fine-Tuning

- Fine-tuning adapts the pre-trained LLM for specific tasks by training it on smaller labeled datasets relevant to the application. This step refines the model's performance for tasks like sentiment analysis or domain-specific text generation[12] [13].
- Techniques include:
  - **Supervised Learning:** Explicitly teaching the model to follow instructions or respond to queries accurately[13] [16].
  - **Few-Shot Learning:** Providing a few examples to improve task-specific performance without extensive retraining[12].

4. Reinforcement Learning

- In some cases, reinforcement learning is used to align the model's outputs with desired behaviors. For example:
  - **Reinforcement Learning with Human Feedback (RLHF):** Human annotations guide the model toward producing high-quality responses while discouraging harmful or irrelevant outputs[16].

5. Scaling and Optimization

- Training large-scale LLMs requires significant computational resources such as GPUs or TPUs.
  - Techniques like data parallelism or model parallelism distribute workloads across multiple devices for efficient training[14].
  - Regularization methods like dropout or early stopping prevent overfitting and improve generalization[14].

Phases Summary

The training process can be summarized into three main phases:

1. **Self-Supervised Learning:** Builds foundational language understanding.
2. **Supervised Learning:** Enhances instruction-following capabilities.
3. **Reinforcement Learning:** Aligns outputs with desired behaviors[13] [16].

Practical Insights

- LLMs are foundational models that can be fine-tuned for various use cases such as chatbots (e.g., ChatGPT), document summarization, or code generation.
- Training LLMs requires careful selection of data sources, computational infrastructure, and optimization techniques to balance complexity with efficiency.

By combining these stages effectively, LLMs achieve remarkable capabilities in understanding and generating human-like text across diverse domains.

⁂

What is a token in the language model?

In the context of large language models (LLMs), a **token** is a fundamental unit of text that the model processes. Tokens can represent a variety of linguistic elements, such as:

- Entire words
- Subwords (e.g., prefixes or suffixes)
- Characters
- Punctuation marks
- Special symbols (e.g., `[CLS]`, `[SEP]`, or `[PAD]`)

Tokenization—the process of breaking down text into tokens—is a critical step in preparing data for LLMs. Each token is assigned a unique numerical ID from the model's vocabulary, enabling the model to process and understand the input text.

How Tokens Work in LLMs

1. **Tokenization Process**:
    - Text is divided into smaller units (tokens) using a tokenizer.
    - The tokenizer maps each token to an integer ID based on the model's vocabulary.
    - For example, the sentence *"I heard a dog bark loudly"* might be tokenized as:

      ```
      ["I", "heard", "a", "dog", "bark", "loudly"]
      ```

      and represented numerically as `[^3_1][^3_2][^3_3][^3_4][^3_5][^3_6]` [18] [19] [20].
2. **Embedding Tokens**:
    - Each token ID is converted into a dense vector representation (embedding) that captures its semantic meaning.
    - These embeddings are processed by the model's architecture (e.g., transformer layers) to understand relationships between tokens.
3. **Output Generation**:
    - During text generation, the model predicts the next token in a sequence by evaluating probabilities for all tokens in its vocabulary.
    - The predicted token is appended to the sequence, and the process repeats iteratively until a stopping condition is met (e.g., reaching an end-of-sequence token) [18] [19] [21].

Types of Tokenization

Different tokenization methods are used depending on the language and application:

1. **Word Tokenization**: Splits text into individual words based on spaces or delimiters.
    - Example: *"The cat runs"* → `["The", "cat", "runs"]`
2. **Character Tokenization**: Breaks text into individual characters.
    - Example: *"cat"* → `["c", "a", "t"]`
3. **Subword Tokenization**: Splits words into smaller units (subwords) to handle unknown or rare words effectively.
    - Example: *"unbelievable"* → `["un", "believ", "able"]`
    - Common techniques include Byte Pair Encoding (BPE) and WordPiece.
4. **Special Tokens**: Used for specific purposes in LLMs, such as:
    - `[CLS]` for classification tasks
    - `[SEP]` for separating sequences
    - `[MASK]` for masked language modeling tasks [18] [20] [22].

Why Tokens Are Important

- **Efficiency**: Breaking text into tokens makes it manageable for models to process large sequences.
- **Flexibility**: Subword tokenization allows models to handle rare or out-of-vocabulary words effectively.
- **Context Understanding**: Tokens enable LLMs to learn patterns and relationships between linguistic elements.

For example, GPT models use subword tokenization like BPE to balance vocabulary size and computational efficiency [18] [20] [22].

In summary, tokens are the building blocks of language models, enabling them to process and generate human-like text efficiently. Proper tokenization ensures that models can handle diverse inputs while maintaining performance and accuracy.

⁂

How to estimate the cost of running SaaS-based and Open Source LLM models?

Estimating the cost of running SaaS-based and open-source Large Language Models (LLMs) involves analyzing multiple factors, including infrastructure, usage patterns, and pricing models. Below is a detailed breakdown of how costs are calculated for both approaches.

SaaS-Based LLMs

SaaS-based LLMs are hosted by providers like OpenAI, Anthropic, or Google. Costs are primarily driven by **API usage**, which is typically priced on a per-token basis.

Cost Estimation Steps

1. **Token-Based Pricing**:
   - SaaS providers charge for input and output tokens processed by the model.
   - Example: OpenAI's GPT-4-turbo costs $0.01 per 1,000 input tokens and $0.03 per 1,000 output tokens[23] [24].
2. **Usage Calculation**:
   - Estimate the number of tokens processed monthly based on user activity.
   - For instance, if a SaaS app has 50,000 users generating 10 responses each (1,000 tokens per response), total monthly usage would be:
     $$\text{Monthly Tokens} = 50,000 \times 10 \times 1,000 = 500,000,000 \text{ tokens}$$
   - For GPT-4-turbo:
     $$\text{Monthly Cost} = \frac{500,000,000}{1,000} \times 0.04 = \$20,000$$
3. **Additional Costs**:
   - Fine-tuning or customizations may incur extra fees.
   - Higher context lengths or premium features increase token costs.

Advantages:

- Minimal infrastructure setup.
- Scalable and predictable pricing for small to medium-scale applications.

Limitations:

- Costs can escalate with high usage.
- Limited control over privacy or customization[23] [25].

Open-Source LLMs

Open-source models like LLaMA 2 or Falcon allow organizations to host models themselves. Costs include **hardware infrastructure**, **engineering labor**, and **maintenance**.

Cost Estimation Steps

1. **Infrastructure Costs**:
   - GPU servers are essential for hosting models. For example:
     - An A100 GPU costs ~$2–$3/hour on AWS[23] [26].

        - Monthly cost for a single A100 (730 hours/month) is approximately $2,000[26].
    - Larger models like LLaMA 2 70B require multiple GPUs, scaling costs significantly[27] [28].
2. **Optimization Techniques**:
    - Use quantized models (e.g., 4-bit) to reduce memory and computational requirements.
    - Tools like vLLM or Text Generation WebUI can optimize inference efficiency[23].
3. **Labor Costs**:
    - Engineers for setup and monitoring add ~$5,000/month to expenses[25].
4. **Example Breakdown**:
    - Hosting LLaMA 2 13B with one A100 GPU:

Component	Cost Estimate
GPU Instance (730 hrs)	~$2,000
Load balancing/autoscaling	$200–$500
Engineering labor	~$5,000
Total	~$7,500/month

5. **Scaling Costs**:
    - Serving millions of users requires additional GPUs and infrastructure (e.g., load balancers), increasing costs proportionally[23] [28].

Advantages:

- Full control over data privacy and customization.
- Potentially cheaper for large-scale deployments with optimized infrastructure.

Limitations:

- High upfront investment in hardware and labor.
- Requires technical expertise for deployment and maintenance[26] [25].

Comparison: SaaS vs Open-Source

Feature	SaaS-Based LLMs	Open-Source LLMs
Cost Structure	Pay-per-token pricing	Infrastructure + labor costs
Scalability	Easy to scale via APIs	Requires additional GPUs/infrastructure
Control	Limited control over customization	Full control over deployment
Privacy	Data handled by third-party	Complete data ownership
Initial Investment	Low	High

Feature	SaaS-Based LLMs	Open-Source LLMs
Monthly Cost Example	$20,000 (GPT-4-turbo for heavy usage)	~$7,500 (LLaMA 2 13B with one GPU)

Key Considerations

1. For small-scale applications or startups with limited technical expertise, SaaS-based solutions are more practical due to their low upfront investment and ease of use.
2. For enterprises requiring privacy or handling large-scale workloads, open-source solutions can be cost-effective in the long run despite higher initial investments.

By carefully analyzing usage patterns and operational needs, organizations can make informed decisions about which approach best suits their requirements.

⁂

Temperature Parameter in Language Models

The **temperature** parameter in large language models (LLMs) controls the randomness and creativity of the model's output by adjusting the probability distribution of token selection during text generation. It determines how "adventurous" or "conservative" the model is when choosing the next word or token.

How Temperature Works

1. **Probability Distribution**:
   - LLMs assign probabilities to all possible tokens based on their training.
   - The temperature modifies these probabilities using the formula:

   $$\sigma(z_i) = \frac{e^{z_i/T}}{\sum_{j=0}^{N} e^{z_j/T}}$$

   Here, T is the temperature value, z_i represents the logits for a token, and $\sigma(z_i)$ is the normalized probability.
2. **Effect of Temperature**:
   - **Low Temperature (<1)**: Sharpens the probability distribution, favoring high-probability tokens. This results in deterministic and predictable outputs[29] [30].
   - **High Temperature (>1)**: Flattens the distribution, increasing the likelihood of selecting less probable tokens. This leads to more diverse and creative responses[30] [31].

Setting the Temperature

The ideal temperature depends on the task at hand:

Temperature Range	Characteristics	Use Cases
Low (0.0–0.5)	Predictable, focused, deterministic	Fact-based tasks like summarization, technical writing, or question answering [32] [33].
Medium (0.6–1.0)	Balanced creativity and coherence	General conversation, customer support, or business writing [34] [33].
High (1.1–2.0)	Creative but less predictable	Storytelling, brainstorming, poetry, or marketing content [32] [33].

Tips for Adjusting Temperature

1. **Start with Defaults**:
   - Many models default to a temperature of 1.0, providing balanced outputs suitable for most applications [35] [36].
2. **Experimentation**:
   - For highly structured tasks requiring accuracy (e.g., legal or medical applications), lower temperatures (e.g., 0.2–0.3) are recommended.
   - For creative tasks (e.g., generating unique ideas), higher temperatures (e.g., 1.2–1.5) can be explored [31] [37].
3. **Monitor Outputs**:
   - Lower temperatures reduce hallucinations but may lead to repetitive responses.
   - Higher temperatures increase diversity but may produce nonsensical or off-topic outputs [32] [31].

Examples

Prompt: *"Describe a sunny day."*

- **Temperature = 0.2**: *"The sun is shining brightly in a clear blue sky."* (Predictable and factual)
- **Temperature = 1.0**: *"Golden sunlight streams through trees as a gentle breeze flows."* (Balanced creativity)
- **Temperature = 1.5**: *"Rays of light dance playfully across emerald leaves, whispering promises of adventure."* (Highly creative) [37].

Key Considerations

- The temperature parameter does not affect accuracy; it influences style and randomness[37].
- For high-stakes applications where reliability is critical, keep the temperature low.
- For creative exploration or brainstorming tasks, higher temperatures can unlock imaginative possibilities.

By understanding and adjusting this parameter effectively, developers can tailor LLM outputs to suit diverse requirements across precision-focused and creativity-driven tasks.

⁂

What are different decoding strategies for picking output tokens?

Decoding strategies are methods used by language models to select the next token in a sequence during text generation. These strategies influence the quality, coherence, and creativity of the generated text. Below is an overview of popular decoding strategies:

Deterministic Decoding Strategies

1. Greedy Search

- **Mechanism**: At each step, the model selects the token with the highest probability.
- **Advantages**:
  - Computationally efficient.
  - Suitable for fact-based tasks requiring predictable outputs.
- **Disadvantages**:
  - May produce repetitive or incomplete text due to lack of exploration of alternative paths[38] [39] [40].
- **Use Cases**: Real-time applications like speech recognition or autocomplete systems.

2. Beam Search

- **Mechanism**: Maintains a fixed number (k) of candidate sequences (beam width) at each step and expands them by considering the top k tokens for each sequence. The sequence with the highest overall score is selected at the end.
- **Advantages**:
  - Explores multiple paths, producing higher-quality text than greedy search.
  - Reduces repetition compared to greedy search.
- **Disadvantages**:

    - Computationally intensive as beam width increases.
    - Tends to produce repetitive and less diverse outputs in open-ended tasks[39] [41] [42].
- **Use Cases**: Machine translation or structured text generation.

Stochastic Decoding Strategies

3. Sampling

- **Mechanism**: Selects tokens randomly based on their probabilities from the softmax distribution.
- **Advantages**:
    - Introduces variability and creativity in text generation.
- **Disadvantages**:
    - Can produce nonsensical or irrelevant outputs if probabilities are not well-controlled[43].
- **Use Cases**: Creative writing or brainstorming tasks.

4. Top-k Sampling

- **Mechanism**: Limits sampling to the top k tokens with the highest probabilities at each step, then selects one token randomly from this subset.
- **Advantages**:
    - Balances diversity and coherence by restricting randomness to high-probability tokens.
- **Disadvantages**:
    - Requires careful tuning of k for optimal results[44] [45].
- **Use Cases**: Applications requiring controlled creativity like content creation.

5. Top-p (Nucleus Sampling)

- **Mechanism**: Dynamically selects tokens whose cumulative probability exceeds a threshold p, ensuring flexibility in the number of tokens considered at each step.
- **Advantages**:
    - Adapts to varying probability distributions, offering greater diversity than top-k.
    - Avoids abrupt cutoff seen in top-k sampling[44] [45].
- **Disadvantages**:
    - Requires tuning p for desired balance between coherence and creativity.
- **Use Cases**: Open-ended text generation such as storytelling or dialogue systems.

Hybrid Approaches

Some applications combine deterministic and stochastic methods for better control over output quality. For example:

- Beam search combined with sampling can introduce randomness while maintaining coherence.

Choosing Decoding Strategies

The choice of decoding strategy depends on task requirements:

Strategy	Best For	Examples
Greedy Search	Predictable, fact-based tasks	Speech recognition, autocomplete
Beam Search	Structured and high-likelihood outputs	Machine translation
Sampling	Creative tasks with high variability	Poetry, brainstorming
Top-k Sampling	Controlled randomness	Content creation
Top-p Sampling	Flexible diversity	Storytelling, dialogue systems

By understanding these strategies, developers can tailor decoding mechanisms to meet specific application needs while balancing coherence, diversity, and computational efficiency.

⁂

What are different ways you can define stopping criteria in large language model?

Stopping criteria in large language models (LLMs) are mechanisms used to control when the model halts text generation. These criteria ensure the output meets desired requirements, such as length, relevance, or coherence, while avoiding excessive computation or irrelevant outputs. Below are various ways to define stopping criteria:

1. Max Tokens

- **Description**: Limits the number of tokens generated by the model.
- **Usage**: Useful for applications requiring concise outputs or avoiding excessive computation.
- **Example**: `max_tokens=100` stops generation after 100 tokens.
- **Advantages**:
    - Simple and predictable.
    - Prevents overly long outputs.
- **Disadvantages**: May truncate responses before reaching logical completion[46] [47].

2. Stop Sequences

- **Description**: Specifies sequences of tokens or phrases that signal the end of generation.
- **Usage**: Commonly used in APIs to stop generation when specific markers (e.g., "End of text") appear in the output.
- **Example**:

  ```
  stop=["\n", "End of text"]
  ```

  Stops generation when a newline or "End of text" is encountered.
- **Advantages**:
  - Ensures logical stopping points based on content.
  - Highly customizable for specific tasks (e.g., chatbot responses).
- **Disadvantages**:
  - Requires careful definition of stop sequences to avoid unintended truncation[46] [48].

3. End-of-Sentence

- **Description**: Stops generation when the model predicts the end of a sentence or a delimiter (e.g., period, exclamation mark).
- **Usage**: Ideal for generating coherent and complete sentences.
- **Example**: Stops at punctuation marks like `.` or `?`.
- **Advantages**:
  - Produces grammatically complete sentences.
- **Disadvantages**:
  - May not work well for tasks requiring multi-sentence outputs[46] [47].

4. Contextual Indicators

- **Description**: Stops generation based on specific contextual cues within the generated text (e.g., detecting certain keywords or topics).
- **Usage**: Implement logic to monitor the content and stop when predefined conditions are met.
- **Example**: Stop if a certain topic (e.g., "error") appears in the output.
- **Advantages**:
  - Enables dynamic control based on content relevance.
- **Disadvantages**:
  - Requires custom implementation and monitoring logic[46].

5. Entropy-Based Stopping

- **Description**: Uses entropy (uncertainty) of token probabilities to decide when to stop. Stops when entropy falls below a threshold, indicating confident predictions.
- **Usage**: Suitable for speculative decoding techniques like AdaEDL1.
- **Example**:

```
if entropy &lt; threshold:
    stop_generation()
```

- **Advantages**:
  - Prevents repetitive or low-information outputs.
- **Disadvantages**:
  - Computationally intensive and requires threshold tuning[46] [47].

6. Length-Based Stopping

- **Description**: Stops generation based on minimum and maximum length constraints for the output.
- **Usage**: Ensures outputs are within a defined range for tasks like summarization or dialogue systems.
- **Example**:

```
min_length=50
max_length=200
```

- **Advantages**:
  - Provides control over output size.
- **Disadvantages**:
  - May produce incomplete responses if limits are too restrictive[46] [47].

7. Custom Stopping Criteria

- **Description**: Define custom logic for stopping by implementing a subclass of stopping criteria tailored to specific needs.
- **Usage**: Useful for advanced applications requiring precise control over generation behavior.
- **Example Implementation in HuggingFace Transformers API**:

```
from transformers import StoppingCriteria, StoppingCriteriaList

class CustomStoppingCriteria(StoppingCriteria):
    def __call__(self, input_ids, scores):
        # Custom logic here
        return condition_met
```

```
stopping_criteria = StoppingCriteriaList([CustomStoppingCriteria()])
```

- **Advantages**:
  - Fully customizable for unique requirements.
- **Disadvantages**:
  - Requires programming expertise and testing [49] [48].

Additional Techniques

Repetition Penalty

Discourages repeated tokens by penalizing their likelihood during generation. Helps avoid repetitive loops in open-ended tasks [50].

Constrained Beam Search

Restricts generation paths to include or exclude specific subsequences using constraints during beam search. Useful for ensuring specific content in outputs [51].

Summary Table

Stopping Criteria	Best For	Advantages	Disadvantages
Max Tokens	Concise outputs	Simple and predictable	May truncate logical completions
Stop Sequences	Controlled task-specific stopping	Highly customizable	Requires careful sequence definition
End-of-Sentence	Coherent sentences	Grammatically complete outputs	Limited multi-sentence control
Contextual Indicators	Dynamic content-based stopping	Flexible and context-aware	Complex implementation
Entropy-Based Stopping	Preventing repetitive/low-information text	Reduces redundancy	Computationally expensive
Length-Based Stopping	Summarization/dialogue systems	Control over output size	Risk of incomplete responses
Custom Criteria	Advanced applications	Fully customizable	Requires programming expertise

By combining these methods effectively, developers can tailor LLM outputs to meet diverse application needs while optimizing performance and relevance.

⁂

How to Use Stop Sequences in LLMs

Stop sequences are a parameter used in large language models (LLMs) to control when the model halts text generation. By specifying predefined strings or tokens, stop sequences ensure that the output stops at logical endpoints, improving response structure, controlling length, and reducing unnecessary token generation. Below is a detailed guide on how to use stop sequences effectively.

What Are Stop Sequences?

A stop sequence is a specific string or set of strings that signals the model to terminate text generation. When the model encounters a stop sequence during generation, it halts immediately, ensuring concise and controlled outputs.

Examples of Stop Sequences:

- `</output>`
- `"END"`
- Newline characters (`\n`)
- Punctuation marks (e.g., `.`)

Steps to Implement Stop Sequences

1. Identify the Stop Sequence

- Choose a string or token that marks the logical endpoint for your task.
- Example:
  - If generating structured JSON responses, use `}` as the stop sequence.
  - For FAQs, use phrases like `"I hope this helps"`.

2. Set the Stop Sequence in the API

Most LLM APIs allow you to specify stop sequences as part of the request parameters.

OpenAI API Example:

```
import openai

response = openai.ChatCompletion.create(
    model="gpt-4",
    messages=[
        {"role": "user", "content": "Write a short paragraph about roses."}
    ],
    stop=[".&lt;/output&gt;", "\n"]
)
```

```
print(response.choices[^8_0].message["content"])
```

Here, the model will halt generation upon encountering either `.</output>` or a newline character.

3. Test and Adjust

- Generate sample outputs and verify if the stop sequence is working as intended.
- If the model stops too early or fails to recognize the sequence, refine your prompt or add additional stop sequences.

Applications of Stop Sequences

1. **Structured Outputs**:
    - Ensure responses follow formats like JSON or XML by stopping at logical endpoints.
    - Example: Use `</output>` for XML responses to prevent extra text generation [52] [53].
2. **Cost Management**:
    - Limit token usage by stopping generation early, reducing API costs (e.g., SaaS models charge per token) [52] [53].
3. **Concise Responses**:
    - Prevent excessive or irrelevant text generation in tasks like summarization or Q&A [54] [55].
4. **List Generation**:
    - Use stop sequences for organized lists by stopping at predefined markers (e.g., `"==="`) [56] [53].

Best Practices for Using Stop Sequences

1. **Choose Unique Strings**:
    - Use strings unlikely to appear naturally in generated text (e.g., `\a`, `<|endoftext|>`) [57] [58].
2. **Combine with Prompt Engineering**:
    - Design prompts that encourage outputs aligned with your stop sequence to ensure proper termination [55] [59].
3. **Set Multiple Stop Sequences**:
    - Add fallback options by specifying multiple sequences in APIs (e.g., `[“END”, “STOP”]`) [52] [59].
4. **Test Thoroughly**:

- Validate outputs across diverse prompts to ensure reliability and adjust sequences as needed [55] [53].

Limitations

- If the model does not encounter a specified stop sequence during generation, it may continue producing irrelevant text.
- Overuse of stop sequences can lead to abrupt terminations or errors if not carefully tuned [60] [54].
- Requires careful prompt design to ensure the sequence appears naturally in generated outputs.

Summary Table

Feature	Description	Example Usage
Structured Outputs	Stops at logical endpoints for formats like JSON	Use `}` for JSON responses
Cost Reduction	Limits unnecessary token generation	Stops after `"END"`
Concise Responses	Prevents excessive output	Stops at newline (`\n`)
List Generation	Ensures organized content	Stops at `"==="`

By using stop sequences strategically, developers can enhance control over LLM outputs while optimizing costs and ensuring structured responses tailored to specific applications.

⁂

Explain the basic structure prompt engineering.

The **basic structure of prompt engineering** involves designing prompts that effectively guide a large language model (LLM) to generate desired outputs. A well-crafted prompt typically consists of four key components: **Instruction**, **Context**, **Input**, and **Output Indicator**. These elements work together to ensure clarity, relevance, and precision in the model's responses.

Key Components of a Prompt

1. Instruction

- **Definition**: The directive or task you want the model to perform.
- **Purpose**: Clearly specifies what the model should do, ensuring it understands the objective.
- **Examples**:
    - *"Summarize the following article in three sentences."*
    - *"Translate this text into French."*

- **Best Practices**:
  - Be concise and specific.
  - Use action verbs like "write," "list," "translate," or "classify."

2. Context

- **Definition**: External information or additional details that help the model generate accurate and relevant responses.
- **Purpose**: Provides background knowledge or constraints to steer the model's output.
- **Examples**:
  - For text classification: *"Here are some examples of positive and negative sentiments."*
  - For retrieval-augmented generation (RAG): *"Use the provided context from a database to answer the question."*
- **Best Practices**:
  - Include relevant examples or domain-specific knowledge when necessary.
  - Avoid overwhelming the model with excessive or irrelevant context.

3. Input

- **Definition**: The user query or data that the model is expected to process.
- **Purpose**: Represents the core information or question that needs a response.
- **Examples**:
  - *"What are the benefits of renewable energy?"*
  - Providing a paragraph for summarization: *"Summarize this text: <<text>>."*
- **Best Practices**:
  - Ensure clarity and avoid ambiguity in phrasing.
  - Format input appropriately for structured tasks (e.g., JSON, tables).

4. Output Indicator

- **Definition**: Specifies the format, style, or type of response expected from the model.
- **Purpose**: Guides the structure of the output to align with user needs.
- **Examples**:
  - *"Provide your answer in bullet points."*
  - *"Output the result as a JSON object."*
- **Best Practices**:
  - Clearly define output constraints (e.g., length, format).

- Use templates for structured responses when applicable.

Complete Prompt Example

Here's how these components come together:

```
Instruction: Summarize the following text in three sentences.

Context: The text is about climate change and its impact on global ecosystems.

Input: Climate change is causing rising temperatures, melting glaciers, and increasing se

Output Indicator: Provide your summary as bullet points.
```

Expected Output:

```
- Rising temperatures are causing glaciers to melt and sea levels to rise.
- Ecosystems are being disrupted globally due to these changes.
- Species extinction and biodiversity loss are major consequences of climate change.
```

Iterative Refinement Process

Prompt engineering often involves refining prompts through an iterative process:

1. **Draft an Initial Prompt**:
   Start with a basic structure based on task requirements.
2. **Test Outputs**:
   Use the LLM to generate responses and evaluate their quality.
3. **Refine Prompt Design**:
   Adjust instructions, add context, or clarify input/output specifications based on evaluation results.
4. **Repeat Until Desired Output Quality is Achieved**:
   Iterate until responses consistently meet expectations across diverse scenarios.

Advanced Techniques

For complex tasks, you can enhance prompts using techniques like:

1. Role-playing instructions (e.g., *"Act as a financial analyst..."*).
2. Few-shot learning by providing examples within the prompt.
3. Constraints like length limits (e.g., *"Explain this in less than 100 words."*).

Summary Table

Component	Purpose	Example
Instruction	Specifies what task to perform	"Summarize this article in three sentences."
Context	Provides external information	"Use examples from climate studies."
Input	Core query or data	"Climate change impacts ecosystems."
Output Indicator	Defines response format	"Provide answers as bullet points."

By understanding and applying these components effectively, you can craft prompts that maximize LLM performance across diverse applications such as content generation, summarization, translation, and more.

⁂

What is In-Context Learning?

In-context learning (ICL) is a technique in large language models (LLMs) where the model adapts to a task by using examples or instructions embedded directly within the input prompt. Unlike traditional machine learning methods, ICL does not involve retraining or fine-tuning the model's parameters. Instead, it leverages the model's pre-trained knowledge to infer patterns and generate outputs based on the provided context during inference time[61] [62] [63].

How In-Context Learning Works

Key Mechanisms

1. **Prompt Engineering**:
    - The user provides a prompt containing task instructions and examples of input-output pairs.
    - Examples serve as demonstrations of the desired behavior or format.
2. **Pattern Recognition**:
    - The model uses its attention mechanisms to identify patterns in the examples, such as structure, tone, or mapping between inputs and outputs[62] [63].
3. **Task Execution**:
    - After processing the prompt, the model generates responses for new inputs by applying the learned patterns from the context[64].

Types of In-Context Learning

1. **Zero-Shot Learning**:
    - No examples are provided; only task instructions guide the model.
    - Example: *"Translate this sentence into French: 'Hello, how are you?'"*
2. **One-Shot Learning**:
    - A single input-output example is included in the prompt.
    - Example: *"Translate this sentence into French: 'Hello, how are you?' → 'Bonjour, comment ça va?' Translate: 'Good morning.'"*
3. **Few-Shot Learning**:
    - Multiple input-output examples are included to demonstrate the task.
    - Example:

    ```
    Translate these sentences into French:
    'Hello, how are you?' → 'Bonjour, comment ça va?'
    'Good morning' → 'Bonjour'
    Translate: 'See you later.'
    ```

Advantages of In-Context Learning

1. **No Parameter Updates**:
    - ICL operates entirely within inference; it does not require retraining or fine-tuning [63] [65].
2. **Flexibility**:
    - Models can adapt to diverse tasks with minimal effort by simply modifying the prompt [66] [63].
3. **Efficiency**:
    - Reduces computational overhead compared to fine-tuning large models for every new task [63] [65].
4. **Applicability**:
    - Works well for tasks with limited labeled data or when rapid prototyping is needed [67] [63].

Limitations of In-Context Learning

1. **Dependence on Prompt Quality**:
    - The effectiveness of ICL heavily relies on well-designed prompts with relevant and high-quality examples [68] [69].
2. **Context Window Constraints**:

- LLMs have a finite context window (e.g., 4,096 tokens for GPT-3), limiting the number of examples that can be included [64] [70].

3. **Transient Knowledge**:
   - The model does not retain information from prompts after inference, making it unsuitable for tasks requiring persistent learning [63] [65].
4. **Performance Variability**:
   - Results can vary depending on how closely the task aligns with patterns learned during pre-training [69] [65].

Applications of In-Context Learning

1. **Text Translation**:
   - Demonstrating translations within prompts enables multilingual text generation.
2. **Question Answering**:
   - Providing examples of question-answer pairs helps models answer similar queries.
3. **Summarization**:
   - Few-shot prompts can guide models to summarize documents effectively.
4. **Classification Tasks**:
   - Examples of labeled data (e.g., sentiment analysis) help models classify new inputs.
5. **Creative Writing**:
   - Using prompts with stylistic examples enables tailored content generation.

Best Practices for Effective In-Context Learning

1. **Use Relevant Examples**:
   - Ensure examples closely match the desired task and output format [68] [71].
2. **Maintain Consistent Formatting**:
   - Uniform formatting improves pattern recognition and response consistency [68] [63].
3. **Optimize Example Order**:
   - Arrange examples logically (e.g., simplest to most complex) or prioritize relevance [68].
4. **Limit Example Count**:
   - Avoid exceeding the model's context window; 2–8 examples are often sufficient for few-shot learning [68] [69].
5. **Experiment Iteratively**:
   - Test and refine prompts to achieve optimal results across diverse tasks [63] [72].

Summary Table

Feature	Description	Example
Zero-Shot Learning	No examples; relies on task instructions	"Summarize this article."
One-Shot Learning	Single example provided	"Translate: 'Hello' → 'Bonjour'"
Few-Shot Learning	Multiple examples provided	"Translate: 'Hello' → 'Bonjour'; ..."

In-context learning is a powerful technique that enables LLMs to perform novel tasks efficiently without retraining, making it highly versatile for real-world applications in NLP and generative AI domains.

⁂

Explain type of prompt engineering

Prompt engineering involves crafting input prompts to guide large language models (LLMs) in generating desired outputs. Various techniques have been developed to improve the quality, accuracy, and relevance of responses. Below are the key types of prompt engineering techniques:

1. Zero-Shot Prompting

- **Description**: Instructs the model to perform a task without providing any examples.
- **Use Case**: Tasks where the model can rely on its pre-trained knowledge.
- **Example**:

  ```
  Classify the sentiment of this text as positive, negative, or neutral: "I think the m
  ```

  Output: Neutral

2. One-Shot Prompting

- **Description**: Provides one example of the desired input-output behavior in the prompt.
- **Use Case**: Tasks requiring guidance on style, tone, or structure.
- **Example**:

  ```
  Translate the following sentence into French:
  "Hello, how are you?" → "Bonjour, comment ça va?"
  Translate: "Good morning."
  ```

  Output: Bonjour

3. Few-Shot Prompting

- **Description**: Includes a few examples (typically 2–4) to demonstrate the task explicitly.
- **Use Case**: Tasks requiring consistency in format or domain-specific knowledge.
- **Example**:

  ```
  Translate these sentences into French:
  - "Hello, how are you?" → "Bonjour, comment ça va?"
  - "Good morning." → "Bonjour."
  Translate: "See you later."
  ```

  Output: À plus tard

4. Chain-of-Thought (CoT) Prompting

- **Description**: Guides the model to break down complex tasks into intermediate reasoning steps before arriving at an answer.
- **Use Case**: Tasks requiring logical reasoning or multi-step problem-solving.
- **Example**:

  ```
  Q: If a train travels at 60 miles per hour for 2 hours, how far does it travel?
  A: First, calculate the distance traveled in one hour. Then multiply by two hours. Th
  ```

5. Tree-of-Thought Prompting

- **Description**: Generalizes CoT prompting by exploring multiple possible reasoning paths using a tree structure.
- **Use Case**: Complex decision-making or problem-solving tasks with multiple possible solutions.
- **Example**:
  - Step 1: List environmental effects of climate change.
  - Step 2: List social effects of climate change.

6. Maieutic Prompting

- **Description**: Prompts the model to generate an explanation for its answer and then iteratively refine it by explaining parts of the explanation.
- **Use Case**: Complex commonsense reasoning or fact-based tasks requiring high accuracy.
- **Example**:

  ```
  Q: Why is the sky blue?
  A: The sky appears blue because blue light is scattered in all directions by gases ir
  ```

7. Least-to-Most Prompting

- **Description**: Breaks down a problem into subproblems and solves them sequentially, ensuring later subproblems benefit from earlier solutions.
- **Use Case**: Tasks requiring step-by-step execution, such as math problems or procedural instructions.
- **Example**:

```
Solve for x in the equation: 2x + 3 = 11.
Step 1: Subtract 3 from both sides → Result: 2x = 8.
Step 2: Divide by 2 → Result: x = 4.
```

8. Self-Refine Prompting

- **Description**: Prompts the model to solve a task, critique its solution, and refine it iteratively until a stopping criterion is met.
- **Use Case**: Content generation tasks like essays or code where iterative improvement is needed.
- **Example**:

```
Write an essay on climate change. Critique it for lack of examples and rewrite with s
```

9. Directional-Stimulus Prompting

- **Description**: Includes specific keywords or cues to guide the model toward desired outputs.
- **Use Case**: Creative tasks like poetry or storytelling where specific themes or tones are needed.
- **Example**:

```
Write a poem about love that includes the words "heart," "passion," and "eternal."
```

10. Generated Knowledge Prompting

- **Description**: Prompts the model to generate relevant facts first and then use those facts to complete the task.
- **Use Case**: Tasks like essay writing or summarization where factual accuracy is crucial.
- **Example**:

```
Generate facts about deforestation:
  - Deforestation contributes to climate change.
  - It leads to loss of biodiversity.

  Write an essay on deforestation using these facts.
```

11. Iterative Prompting

- **Description**: Involves asking follow-up questions after initial responses to refine or clarify outputs further.
- **Use Case**: Conversational AI or tasks requiring detailed exploration of a topic.

Summary Table

Technique	Best For	Example Use Case
Zero-Shot Prompting	General tasks	Sentiment analysis
One-Shot Prompting	Style/tone guidance	Translation
Few-Shot Prompting	Consistency in format	Domain-specific text generation
Chain-of-Thought Prompting	Logical reasoning	Math problems
Tree-of-Thought Prompting	Multi-path decision-making	Environmental impact analysis
Maieutic Prompting	Complex reasoning	Commonsense questions
Least-to-Most Prompting	Step-by-step problem-solving	Procedural tasks
Self-Refine Prompting	Iterative content improvement	Essay writing
Directional-Stimulus	Thematic content creation	Poetry
Generated Knowledge	Fact-based generation	Essays

By leveraging these techniques effectively, prompt engineering can maximize LLM performance across diverse applications such as reasoning, content creation, summarization, and more.

⁂

What are some of the aspect to keep in mind while using few-shots prompting?

When using **few-shot prompting**, there are several aspects to keep in mind to ensure effective and accurate results. Few-shot prompting involves providing a small number of examples within the prompt to guide the model's behavior for specific tasks. Below are key considerations and best practices:

1. Selecting Effective Examples

- **Relevance**: Choose examples that are directly related to the task you want the model to perform. Irrelevant examples can confuse the model and degrade performance[73] [74] [75].
- **Diversity**: Include examples that cover a range of scenarios or edge cases within the task. This helps the model generalize better to unseen inputs[75] [76].

- **Clarity**: Ensure examples are unambiguous, simple, and easy for the model to interpret. Avoid overly complex or convoluted examples[73] [74].

2. Structuring the Prompt

- **Consistency in Format**: Maintain a uniform structure for input-output pairs across all examples. For instance, use a clear delimiter like `Input:` and `Output:` or a Q&A format[74] [76].
- **Order of Examples**: Experiment with example order (e.g., randomizing or prioritizing simpler examples first) to optimize performance[75].
- **Output Specification**: Clearly define the desired output format (e.g., JSON, bullet points, or plain text) within the prompt[76].

3. Number of Examples

- **Optimal Range**: Use 2–5 examples as a starting point. While adding more examples can improve accuracy, exceeding the context window may lead to truncation or inefficiency[75] [77].
- **Avoid Overfitting**: Too many similar examples can cause the model to mimic patterns excessively, reducing its ability to generalize[73] [75].

4. Providing Context

- Include relevant background information or instructions in the prompt if needed. This helps set up the task more effectively, especially for specialized domains like legal or medical applications[74] [75].
- Example:

  ```
  Task: Classify movie reviews as positive, negative, or neutral.
  ```

5. Testing and Refinement

- Test prompts across diverse inputs to evaluate generalization.
- Refine prompts iteratively by adjusting examples, format, or instructions based on observed outputs[73] [75].

6. Avoiding Bias and Overfitting

- Include both positive and negative examples to avoid bias toward majority labels in the prompt[75].
- Ensure examples represent a balanced distribution of possible outputs.

7. Handling Context Window Constraints

- Be mindful of token limits in LLMs (e.g., 4,096 tokens for GPT models). Use concise examples and avoid lengthy prompts that exceed this limit [76] [77].

8. Use Cases for Few-Shot Prompting

Few-shot prompting is particularly useful in scenarios such as:

- **Specialized Domains**: Tasks requiring domain-specific knowledge without extensive labeled data (e.g., legal or technical writing) [75].
- **Content Creation**: Generating structured outputs with consistent style and tone (e.g., product descriptions) [73] [75].
- **Dynamic Applications**: Adapting quickly to new tasks without fine-tuning (e.g., chatbots) [75].

Summary Table

Aspect	Description	Example/Best Practice
Selecting Examples	Relevant, diverse, and clear	Include edge cases and avoid ambiguity
Structuring Prompts	Consistent format and output specification	Use `Input:`/`Output:` delimiters
Number of Examples	2–5 examples recommended	Avoid exceeding context window
Providing Context	Include task instructions	"Classify reviews as positive/negative"
Testing & Refinement	Iterative adjustments based on outputs	Test on diverse inputs
Avoiding Bias	Balance positive/negative examples	Include varied labels

By following these principles, few-shot prompting can significantly enhance performance across diverse applications while minimizing risks like overfitting or bias.

⁂

What are certain strategies to write good prompt?

Writing effective prompts for large language models (LLMs) is crucial to achieving high-quality, relevant, and accurate outputs. Below are strategies to craft good prompts, based on best practices and techniques:

1. Understand the Objective

- **Why**: Clearly define what you want the model to accomplish—whether it's generating creative content, answering a question, or solving a problem.
- **How**: Formulate the prompt to reflect this objective explicitly.
- **Example**:

```
Write a professional email inviting a colleague to a meeting next week.
```

2. Be Clear and Concise

- **Why**: Ambiguous or overly complex prompts can confuse the model and produce suboptimal results.
- **How**: Use straightforward language and avoid unnecessary details.
- **Example**:

```
Summarize the following article in three sentences.
```

3. Provide Context

- **Why**: LLMs perform better when given background information or specific instructions that set the stage for the task.
- **How**: Include relevant details about the task, audience, or scenario.
- **Example**:

```
You are a customer service representative. Respond to this customer complaint about c
```

4. Use Examples (Few-Shot Prompting)

- **Why**: Demonstrating input-output pairs helps the model understand the desired task format.
- **How**: Provide one or more examples before presenting the actual task.
- **Example**:

```
Translate these sentences into French:
- "Hello, how are you?" → "Bonjour, comment ça va?"
- "Good morning." → "Bonjour."
Translate: "See you later."
```

5. Specify Output Format

- **Why**: Defining the desired structure ensures the output meets your expectations (e.g., bullet points, JSON, paragraphs).
- **How**: Include explicit instructions for formatting.
- **Example**:

```
List three benefits of renewable energy in bullet points.
```

6. Tailor Tone and Style

- **Why**: Adjusting tone ensures the output aligns with your audience or purpose (e.g., formal, casual, technical).
- **How**: Specify tone requirements in the prompt.
- **Example**:

```
Write an engaging blog post about sustainable living in a friendly and conversational
```

7. Break Down Complex Tasks

- **Why**: Simplifying tasks into smaller steps improves accuracy and coherence for multi-step problems.
- **How**: Use Chain-of-Thought (CoT) prompting or guide step-by-step reasoning.
- **Example**:

```
Solve for x in the equation: 2x + 3 = 11. First, subtract 3 from both sides. Then div
```

8. Experiment and Iterate

- **Why**: The first version of a prompt may not yield optimal results; refining it can improve performance.
- **How**: Test different phrasings, add/remove details, or reorder instructions to find what works best.

9. Avoid Ambiguity

- **Why**: Vague prompts can lead to irrelevant or incomplete outputs.
- **How**: Use precise language and avoid conflicting instructions.
- **Example (Poor)**:

```
Write about renewable energy quickly but also be detailed.
```

Better:

```
Write a detailed overview of renewable energy in under 200 words.
```

10. Use Role-Based Instructions

- **Why**: Assigning roles helps the model adopt specific perspectives or tones (e.g., teacher, developer).
- **How**: Begin the prompt with role-playing instructions.
- **Example**:

```
You are a software engineer. Explain how APIs work to a non-technical audience.
```

11. Set Length Constraints

- **Why**: Controlling response length prevents overly verbose or insufficient outputs.
- **How**: Specify word count, number of sentences, or token limits in the prompt.
- **Example**:

```
Summarize this article in no more than three sentences.
```

12. Combine Multiple Tasks

- **Why**: Combining tasks can save time when you need comprehensive outputs in one response.
- **How**: Use clear separators for each task within the prompt.
- **Example**:

```
Explain renewable energy in simple terms. Then list three examples of renewable energ
```

Summary Table

Strategy	Why It Helps	Example
Understand Objective	Aligns output with goals	"Write an email inviting someone to lunch."
Be Clear and Concise	Avoids confusion	"Summarize this text in three sentences."
Provide Context	Adds relevance	"You are an HR manager writing an onboarding email."
Use Examples	Demonstrates desired format	Few-shot prompting for translation tasks

Strategy	Why It Helps	Example
Specify Output Format	Ensures structured responses	"List benefits as bullet points."
Tailor Tone	Matches audience needs	"Write formally about climate change."
Break Down Complex Tasks	Improves reasoning	Chain-of-thought prompting
Experiment and Iterate	Refines prompt quality	Test different phrasings
Avoid Ambiguity	Reduces irrelevant responses	"Explain APIs clearly in under two paragraphs."
Role-Based Instructions	Guides tone and perspective	"You are a teacher explaining photosynthesis."
Set Length Constraints	Controls verbosity	"Write a summary under three sentences."
Combine Multiple Tasks	Saves time	"Explain X; then list examples of Y."

By combining these strategies effectively, you can craft prompts that maximize LLM performance while ensuring relevance, clarity, and precision across diverse applications.

⁂

What is Hallucination in AI?

Hallucination in AI refers to a phenomenon where large language models (LLMs) generate outputs that are inaccurate, nonsensical, or fabricated, despite appearing plausible and confident. These hallucinations can include:

- False facts or numbers.
- Non-existent citations or research papers.
- Contradictory or irrelevant information.

For example, an LLM might confidently list papers by Albert Einstein published after 1960, even though Einstein died in 1955[78] [79] [80].

Hallucinations occur due to factors such as:

- **Training Data Issues**: Biases, inaccuracies, or gaps in the training data.
- **Model Overconfidence**: Overgeneralization based on statistical correlations rather than factual grounding.
- **Prompt Ambiguity**: Vague or poorly designed prompts that give the model excessive creative freedom[78] [81] [82].

How Can Prompt Engineering Control Hallucinations?

Prompt engineering is a powerful method to mitigate hallucinations by carefully designing inputs that guide LLMs toward accurate and reliable outputs. Below are strategies to reduce hallucinations using prompt engineering:

1. Be Explicit and Context-Rich

- **Why It Works**: Ambiguous prompts allow the model to extrapolate freely, increasing the likelihood of hallucinations. Providing clear instructions narrows the scope of generation.
- **Example**:
  - Weak Prompt: "*Tell me about string theory.*"
  - Improved Prompt: "*Give a short, accurate summary of string theory suitable for high school students and mention two real physicists associated with it.*" [80].

2. Use Role Prompting

- **Why It Works**: Assigning a role (e.g., "You are a physics professor") sets expectations for tone and factual accuracy, reducing errors.
- **Example**:
  - Prompt: "*You are a historian. Provide an accurate summary of the events leading up to World War I.*" [80].

3. Incorporate Retrieval-Augmented Generation (RAG)

- **Why It Works**: Directing the model to use external sources ensures outputs are grounded in factual data rather than relying solely on internal knowledge.
- **Example**:
  - Without RAG: "*Summarize the company's AI products.*"
  - With RAG: "*Based on the provided company documentation, summarize the AI product line.*" [83] [84].

4. Chain-of-Thought Prompting

- **Why It Works**: Asking the model to reason step-by-step reduces errors by encouraging logical progression rather than jumping to conclusions.
- **Example**:
  - Prompt: "*Explain why the sky is blue step by step, including references to Rayleigh scattering.*" [85] [80].

5. Few-Shot Examples

- **Why It Works**: Providing examples demonstrates the desired output format and accuracy level, helping the model mimic correct responses.
- **Example**:

```
Example 1: "What is photosynthesis?" → "Photosynthesis is the process by which plant
Example 2: "What is gravity?" → "Gravity is a force that attracts objects toward eac
Now answer: "What is relativity?"
```

6. Use "According to..." Prompting

- **Why It Works**: Encouraging responses based on specific sources grounds outputs in factual data and reduces fabrication.
- **Example**:
    - Prompt: "*What part of the brain is responsible for long-term memory, according to Wikipedia?*"[85].

7. Limit Output Scope

- **Why It Works**: Constraining response length or limiting options reduces opportunities for hallucination.
- **Example**:
    - Unconstrained Prompt: "*Write a detailed report on quantum mechanics.*"
    - Constrained Prompt: "*In fewer than 100 words, explain quantum entanglement.*"[84][86].

8. Reflective Prompting

- **Why It Works**: Asking the model to review its output encourages self-correction and validation of responses.
- **Example**:
    - Prompt: "*Explain photosynthesis. Reflect on your explanation and ensure it covers all key steps accurately.*"[86].

9. Avoid Ambiguous Prompts

- **Why It Works**: Simplifying language reduces misinterpretation and ensures clarity.
- **Example**:
    - Poor Prompt: "*Expound on fiscal policy ramifications.*"
    - Improved Prompt: "*Explain how recent tax policy changes affect businesses.*"[86].

10. Use Entropy Thresholding

- **Why It Works**: Reducing randomness in token selection lowers the likelihood of generating inaccurate or overly creative outputs.
- **Implementation**:
    - Lower temperature settings (e.g., $ T = 0.2 $) for deterministic responses[87] [86].

Summary Table

Strategy	Why It Helps	Example
Explicit Prompts	Reduces ambiguity	"Summarize NASA's Mars missions using official reports." [80]
Role-Based Prompts	Guides tone and accuracy	"You are a physics professor explaining relativity." [80]
Retrieval-Augmented Prompts	Grounds responses in external data	"Based on company documentation..." [83]
Chain-of-Thought Reasoning	Encourages logical progression	"Explain why water boils step-by-step." [85]
Few-Shot Examples	Demonstrates desired output format	Provide input-output pairs for guidance. [80]
“According to...” Prompts	Grounds answers in trusted sources	"According to Wikipedia..." [85]
Limited Output Scope	Reduces off-topic responses	"Explain quantum entanglement in <100 words." [84]
Reflective Prompts	Encourages self-validation	"Reflect on your explanation." [86]
Avoid Ambiguity	Ensures clarity	Use simple language like "Explain tax changes." [86]
Entropy Thresholding	Reduces randomness	Lower temperature settings. [87] [86]

By combining these strategies effectively, prompt engineering can significantly reduce hallucinations in LLMs while improving accuracy and user trust across diverse applications.

⁂

How to improve the reasoning ability of LLM through prompt engineering?

Improving the reasoning ability of large language models (LLMs) through prompt engineering involves crafting prompts that encourage logical, step-by-step thinking and structured problem-solving. Since LLMs rely on patterns learned during training, specific techniques can guide them to reason more effectively. Below are strategies to enhance reasoning in LLMs using prompt engineering:

1. Chain-of-Thought (CoT) Prompting

- **Description**: Ask the model to break down a problem into intermediate steps before arriving at a final answer.
- **Why It Works**: Encourages step-by-step reasoning, reducing errors in complex tasks.
- **Example**:

```
Q: If a train travels at 60 miles per hour for 2 hours, how far does it travel?
A: First, calculate the distance traveled in one hour. Then multiply by two hours. Th
```

2. Least-to-Most Prompting

- **Description**: Break down a complex task into smaller subproblems and solve them sequentially.
- **Why It Works**: Simplifies reasoning by focusing on one aspect of the problem at a time.
- **Example**:

```
Solve for x in the equation: 2x + 3 = 11.
Step 1: Subtract 3 from both sides → Result: 2x = 8.
Step 2: Divide by 2 → Result: x = 4.
```

3. Role-Based Prompting

- **Description**: Assign a role to the model (e.g., teacher, scientist) to encourage domain-specific reasoning or structured thinking.
- **Why It Works**: Helps the model adopt a logical perspective based on the assigned role.
- **Example**:

```
You are a math teacher. Explain how to calculate the area of a circle step by step.
```

4. Few-Shot Prompting

- **Description**: Provide examples of input-output pairs that demonstrate logical reasoning for similar tasks.
- **Why It Works**: Helps the model learn patterns from examples and apply them to new inputs.
- **Example**:

```
Example 1:
Q: What is the sum of 12 and 15?
A: Add the two numbers together. The result is 27.

Example 2:
Q: What is the sum of 23 and 34?
```

```
A: Add the two numbers together. The result is 57.

Now answer:
Q: What is the sum of 45 and 67?
```

5. Reflective Prompting

- **Description**: Ask the model to critique or validate its own output before finalizing an answer.
- **Why It Works**: Encourages self-checking and refinement, reducing errors in reasoning.
- **Example**:

```
Explain why water boils at lower temperatures at higher altitudes. Reflect on your ex
```

6. Ask "Why" or "How" Questions

- **Description**: Frame prompts as "why" or "how" questions to encourage deeper reasoning rather than surface-level responses.
- **Why It Works**: Forces the model to generate explanations rather than just answers.
- **Example**:

```
Why does adding salt lower the freezing point of water?
```

7. Use Explicit Instructions

- **Description**: Clearly instruct the model to reason step-by-step or explain its thought process explicitly.
- **Why It Works**: Directs the model to focus on logical progression rather than jumping to conclusions.
- **Example**:

```
Solve this problem step by step:
If you have $50 and spend $20 on groceries, how much money do you have left?
```

8. Tree-of-Thought Prompting

- **Description**: Encourage exploration of multiple reasoning paths by asking the model to consider alternative approaches or solutions before deciding on one.
- **Why It Works**: Promotes comprehensive reasoning by evaluating different possibilities.
- **Example**:

```
List three possible reasons why renewable energy adoption is increasing globally. The
```

9. Use Analogies

- **Description**: Ask the model to explain concepts using analogies or comparisons, which require logical mapping between ideas.
- **Why It Works**: Encourages abstract thinking and deeper understanding of relationships between concepts.
- **Example**:

```
Explain how an electric circuit works using an analogy with water flowing through pip
```

10. Iterative Questioning

- **Description**: Break down complex questions into smaller, related subquestions that build toward solving the main problem.
- **Why It Works**: Simplifies complex reasoning tasks by addressing them incrementally.
- **Example**:
  - Main Question:

    ```
    How does climate change affect biodiversity?
    ```

  - Subquestions:

    ```
    - What is climate change?
    - How does temperature rise impact ecosystems?
    - Why do species face extinction as ecosystems change?
    ```

Summary Table

Strategy	Why It Helps	Example
Chain-of-Thought	Encourages step-by-step reasoning	"Explain why X happens step by step."
Least-to-Most	Solves complex problems incrementally	Break tasks into smaller subproblems.
Role-Based Prompting	Adopts logical perspectives	"You are a scientist explaining Y."
Few-Shot Prompting	Demonstrates desired reasoning patterns	Provide input-output examples
Reflective Prompting	Promotes self-checking	"Reflect on your answer for accuracy."
"Why"/"How" Questions	Encourages deeper explanations	"Why does X happen?"
Explicit Instructions	Directs logical progression	"Solve this problem step by step."
Tree-of-Thought	Explores multiple solutions	"List three reasons for X."
Use Analogies	Promotes abstract thinking	"Explain X using an analogy."

Strategy	Why It Helps	Example
Iterative Questioning	Simplifies complex problems	Break main question into subquestions.

By applying these strategies, you can significantly enhance an LLM's ability to reason logically, solve problems systematically, and produce more accurate and coherent outputs across diverse applications such as mathematics, science, decision-making, and more.

How to improve LLM reasoning if your COT prompt fails?

If your **Chain-of-Thought (CoT)** prompt fails to improve reasoning in large language models (LLMs), there are several alternative strategies and complementary techniques you can employ to enhance the model's reasoning capabilities. These approaches address common issues such as shallow reasoning, ambiguity, and lack of domain knowledge.

Why CoT Prompting Might Fail

CoT prompting can fail due to:

1. **Over-reliance on learned patterns**: LLMs mimic reasoning templates rather than truly understanding logic.
2. **Ambiguous or under-specified prompts**: Vague instructions lead to poor reasoning.
3. **Complex or multi-modal tasks**: CoT may struggle with problems requiring external tools or deep domain knowledge.
4. **Prompt fatigue**: Long or convoluted prompts degrade performance.
5. **Limitations of smaller models**: Insufficient capacity for reasoning depth.

Strategies to Improve Reasoning When CoT Fails

1. Self-Consistency Decoding

- **Description**: Generate multiple CoT outputs and take the majority vote from these generations as the final answer.
- **Why It Works**: Filters out hallucinations and incorrect reasoning paths by leveraging diverse outputs.
- **Implementation**:

```
answers = [llm(prompt, temperature=0.7) for _ in range(10)]
final_answer = most_common_final_answer(answers)
```

- **Use Case**: Math problems, logic puzzles, or factual queries where consistency is critical[88] [89].

2. ReAct Prompting (Reason + Act)

- **Description**: Combine reasoning steps with external tool usage (e.g., calculators, knowledge bases).
- **Why It Works**: Grounds reasoning in factual data and allows the model to interleave logical steps with actions.
- **Example Prompt**:

```
Question: What is the capital of the country that borders both Germany and Spain?
Thought: Let me look at the map. Which countries border Germany and Spain?
Action: Search["countries that border Germany and Spain"]
```

- **Use Case**: Tasks requiring external data or multi-modal reasoning [88] [90].

3. Few-Shot CoT Prompting

- **Description**: Provide examples of input-output pairs demonstrating step-by-step reasoning.
- **Why It Works**: Helps the model mimic desired reasoning structures by learning from examples.
- **Example Prompt**:

```
Q: John has 3 apples, buys 2, eats 1. How many apples now?
A: John starts with 3. He buys 2 → 3 + 2 = 5. He eats 1 → 5 - 1 = 4 apples.

Q: Sarah has 10 pencils. She gives away 4, then buys 3 more. How many now?
A:
```

- **Use Case**: Arithmetic problems, logical deductions [90] [91].

4. Decomposition Prompting

- **Description**: Break down complex tasks into smaller subtasks and solve them sequentially.
- **Why It Works**: Simplifies reasoning by focusing on one aspect of the problem at a time.
- **Example Prompt**:

```
Task: Solve for x in the equation 2x + 3 = 11.

Step 1: Subtract 3 from both sides → Result: 2x = 8.

Step 2: Divide by 2 → Result: x = 4.
```

- **Use Case**: Multi-step problems like math equations or procedural workflows [90] [91].

5. Tree-of-Thought (ToT) Prompting

- **Description**: Explore multiple reasoning paths by asking the model to consider alternative approaches before deciding on a solution.
- **Why It Works**: Encourages comprehensive exploration of possibilities and verification of answers.
- **Example Prompt**:

  ```
  List three possible reasons why renewable energy adoption is increasing globally. The
  ```

- **Use Case**: Decision-making tasks or open-ended questions[88] [90].

6. Reflective Prompting

- **Description**: Ask the model to critique or validate its own output before finalizing an answer.
- **Why It Works**: Encourages self-checking and refinement, reducing errors in reasoning.
- **Example Prompt**:

  ```
  Explain photosynthesis step by step. Reflect on your explanation and ensure it covers
  ```

- **Use Case**: Scientific explanations or logical proofs[92].

7. Decoding Modifications

- **Description**: Alter decoding strategies (e.g., top-k sampling) to reveal alternative reasoning paths inherent in the model's predictions.
- **Why It Works**: Uncovers latent CoT paths that may be obscured by greedy decoding techniques.
- **Implementation Example**:
  - Use top-k sampling to explore diverse outputs and identify reliable CoT paths[93] [94].
  - Combine confidence metrics with decoding adjustments.

8. Role-Based Prompting

- **Description**: Assign a specific role to the model (e.g., "You are a PhD-level mathematician").
- **Why It Works**: Frames the task with expectations for logical depth and accuracy.
- **Example Prompt**:

  ```
  You are a historian specializing in World War II events. Explain the causes of the wa
  ```

- **Use Case**: Domain-specific tasks requiring expertise[90].

Summary Table

Strategy	Why It Helps	Example Use Case
Self-Consistency Decoding	Filters hallucinations via majority vote	Math problems, factual queries
ReAct Prompting	Combines reasoning with tool use	Multi-modal tasks
Few-Shot CoT Prompting	Demonstrates desired reasoning patterns	Arithmetic, logical deductions
Decomposition Prompting	Simplifies complex tasks	Multi-step workflows
Tree-of-Thought Prompting	Explores multiple solutions	Decision-making
Reflective Prompting	Encourages self-checking	Scientific explanations
Decoding Modifications	Reveals latent CoT paths	Diverse outputs
Role-Based Prompting	Frames task expectations	Domain-specific tasks

Final Thoughts

When CoT prompting fails, it signals an opportunity to rethink your approach rather than abandoning it altogether. Combining techniques like self-consistency decoding, decomposition prompting, and reflective feedback can significantly enhance LLM reasoning performance across diverse applications[88] [90] [91].

⁂

For all the questions asked provide a nice headline for them and give me a nicely formatted document with questions and answers along with the system instructions at the start in a markdown format

Here is the requested document in **Markdown format**, including a headline for each question, nicely formatted questions and answers, and the system instructions at the beginning.

Interview Preparation: Advanced Topics in LLMs

System Instructions:
You are preparing for interviews for roles in multimodal search, search and recommendations, LLM fine-tuning, and AI research at leading companies such as Google DeepMind, Microsoft, Anthropic, Mistral, Amazon, and Meta. These roles involve working on cutting-edge technologies in LLMs, including fine-tuning pre-trained models, optimizing performance for domain-specific tasks, designing robust search and recommendation systems, and advancing multimodal AI capabilities.

This document contains detailed answers to key technical and research-oriented questions relevant to these roles. The focus is on practical insights that align with the expectations of top-tier companies.

1. What is the difference between Predictive/Discriminative AI and Generative AI?

Headline: Understanding Predictive/Discriminative AI vs. Generative AI

Predictive/Discriminative AI focuses on distinguishing between predefined classes or predicting outcomes by modeling the conditional probability $P(Y|X)$. It excels in tasks like classification (e.g., spam detection) or regression (e.g., predicting stock prices). Examples include logistic regression and neural networks for image recognition.

Generative AI learns the joint probability $P(X,Y)$, enabling it to generate new data resembling the training distribution. It excels in creative tasks like text generation (e.g., GPT models), image synthesis (e.g., DALL-E), or audio creation. It is computationally intensive but highly versatile.

Feature	Predictive/Discriminative AI	Generative AI	
Purpose	Predicts outcomes or classifies data into categories.	Creates new data instances resembling the original dataset.	
Data Modeling	Models $ P(Y	X) $ to find decision boundaries.	Models $ P(X,Y) $ to learn data distributions.
Applications	Fraud detection, sentiment analysis, facial recognition.	Content generation, drug discovery, synthetic data creation.	

2. What is an LLM, and how are LLMs trained?

Headline: What Are Large Language Models (LLMs), and How Are They Trained?

A **Large Language Model (LLM)** is a type of deep learning model designed for natural language processing tasks. It uses billions of parameters to understand and generate human-like text. Training an LLM involves:

1. **Data Collection & Preprocessing**: Text from diverse sources is tokenized into smaller units (e.g., subwords).
2. **Pre-Training**: The model learns statistical patterns by predicting masked words or the next word in a sequence.
3. **Fine-Tuning**: The pre-trained model is adapted to specific tasks using supervised learning or reinforcement learning.
4. **Optimization**: Techniques like distributed training on GPUs/TPUs improve scalability.

LLMs are foundational models that can be fine-tuned for domain-specific applications like chatbots or summarization systems.

3. What is a token in a language model?

Headline: Understanding Tokens in Language Models

A **token** is the basic unit of text that a language model processes. It can represent:

- Words: "cat"
- Subwords: "un-", "believable"
- Characters: "c", "a", "t"
- Special symbols: `[CLS]`, `[MASK]`

Tokenization breaks input text into tokens using techniques like Byte Pair Encoding (BPE). Each token is mapped to an ID from the vocabulary and processed by the model's architecture.

Example:

```
Input Text: "I love cats."
Tokens: ["I", "love", "cats", "."]
```

Tokens enable efficient processing of text while handling rare or unseen words effectively.

4. How to estimate the cost of running SaaS-based and Open Source LLM models?

Headline: Estimating Costs of Running SaaS-Based vs Open Source LLMs

SaaS-Based LLMs:

- Cost depends on API usage (e.g., OpenAI charges per token).
- Example:
  ```
  Monthly Tokens = 50,000 users × 10 responses × 1,000 tokens = 500M tokens
  Cost = $0.04 per 1K tokens → $20,000/month
  ```

Open Source LLMs:

- Costs include infrastructure (GPUs), labor, and maintenance.
- Example:
  ```
  Hosting LLaMA 2 with one A100 GPU costs ~$7,500/month.
  ```

Feature	SaaS-Based LLMs	Open Source LLMs
Cost Structure	Pay-per-token pricing	Infrastructure + labor costs
Scalability	Easy via APIs	Requires additional GPUs
Privacy	Data handled by third-party	Full control over data

5. Explain the Temperature parameter and how to set it.

Headline: Controlling Creativity with the Temperature Parameter

The temperature parameter controls randomness in token selection during text generation:

- Low Temperature (<1): Focused and deterministic outputs.
- High Temperature (>1): Creative but less predictable outputs.

Example:

```
Prompt: "Describe a sunny day."
- Temp = 0.2 → "The sun shines brightly."
- Temp = 1.0 → "Golden sunlight streams through trees."
- Temp = 1.5 → "Rays dance playfully across emerald leaves."
```

Use low temperatures for factual tasks and high temperatures for creative tasks.

6. What are different decoding strategies for picking output tokens?

Headline: Decoding Strategies in Language Models

Decoding strategies determine how tokens are selected during generation:

1. **Greedy Search**: Selects the highest-probability token at each step.
2. **Beam Search**: Explores multiple paths but may produce repetitive outputs.
3. **Top-k Sampling**: Samples from the top-k highest-probability tokens.
4. **Top-p Sampling (Nucleus Sampling)**: Samples from tokens whose cumulative probability exceeds $ p $.

Example:

```
Prompt: "Write a story about space."
Top-p Sampling → Diverse yet coherent stories.
```

7. What is hallucination, and how can it be controlled using prompt engineering?

Headline: Controlling Hallucinations in Large Language Models

Hallucination occurs when an LLM generates false or nonsensical outputs confidently. To mitigate hallucinations:

1. Use explicit prompts with clear instructions.
2. Provide context-rich inputs.
3. Use “According to…” prompts to ground responses in trusted sources.
4. Limit output scope to avoid excessive creativity.

Example:

```
Prompt: "According to NASA's website, what causes eclipses?"
```

8. How to improve the reasoning ability of LLM through prompt engineering?

Headline: Enhancing Reasoning Ability in LLMs

Strategies include:

1. Chain-of-Thought Prompting: Break problems into intermediate steps.
2. Few-Shot Prompting: Provide examples demonstrating reasoning patterns.
3. Reflective Prompting: Ask the model to critique its own answer.
4. Tree-of-Thought Prompting: Explore multiple reasoning paths before deciding.

Example:

```
Q: Solve for x in 2x + 3 = 11.
A:
Step 1: Subtract 3 → Result: 2x = 8.
Step 2: Divide by 2 → Result: x = 4.
```

9. How to improve LLM reasoning if your CoT prompt fails?

Headline: Alternatives When Chain-of-Thought Prompting Fails

If CoT fails:

1. Use Self-Consistency Decoding to filter out errors via majority voting.
2. Apply ReAct Prompting (Reason + Act) for external tool integration.
3. Decompose complex tasks into subtasks using Decomposition Prompting.

4. Use Reflective Prompting for self-validation.

Example:

```
Q: Why does salt lower water’s freezing point?
A: Salt disrupts water's molecular structure, reducing freezing temperature.
Reflect on this explanation for accuracy.
```

This document provides comprehensive answers with practical insights into advanced topics related to large language models (LLMs). Use these strategies effectively during your interview preparation!

1. https://www.restack.io/p/generative-ai-answer-vs-discriminative-ai-vs-predictive-ai-cat-ai
2. https://www.ibm.com/think/topics/predictive-ai
3. https://www.miquido.com/ai-glossary/discriminative-ai/
4. https://olibr.com/blog/generative-ai-vs-discriminative-ai-whats-the-key-difference/
5. https://business.canon.com.au/insights/what-is-the-difference-between-generative-ai-and-discriminative-ai
6. https://www.nnlm.gov/guides/data-thesaurus/generative-artificial-intelligence
7. https://www.coursera.org/articles/what-is-generative-ai
8. https://www.restack.io/p/generative-ai-answer-vs-predictive-ai-vs-discriminative-ai-cat-ai
9. https://www.plainconcepts.com/discriminative-ai-vs-generative-ai/
10. https://www.techtarget.com/whatis/definition/large-language-model-LLM
11. https://www.spiceworks.com/tech/artificial-intelligence/articles/what-is-llm/
12. https://aws.amazon.com/what-is/large-language-model/
13. https://snorkel.ai/blog/large-language-model-training-three-phases-shape-llm-training/
14. https://www.nitorinfotech.com/blog/training-large-language-models-llms-techniques-and-best-practices/
15. https://www.growthloop.com/university/article/llm
16. https://www.alexanderthamm.com/en/blog/an-introduction-to-llm-training/
17. https://www.elastic.co/what-is/large-language-models
18. https://learn.microsoft.com/en-us/dotnet/ai/conceptual/understanding-tokens
19. https://docs.mistral.ai/guides/tokenization/
20. https://www.linkedin.com/pulse/how-understand-tokens-ai-large-language-models-open-ai-gpt-news-uagjc
21. https://seantrott.substack.com/p/tokenization-in-large-language-models
22. https://www.coursera.org/articles/tokenization-nlp
23. https://www.metriccoders.com/post/how-to-estimate-the-cost-of-running-saas-based-vs-open-source-llm-models
24. https://www.tensorops.ai/post/understanding-the-cost-of-large-language-models-llms
25. https://www.linkedin.com/pulse/tco-calculator-evaluate-cost-in-house-llm-deployment-vs-yugank-aman-sr6hf

26. https://www.e2enetworks.com/blog/why-self-hosting-small-llms-are-cheaper-than-gpt-4-a-breakdown
27. https://www.finops.org/wg/cost-estimation-of-ai-workloads/
28. https://www.linkedin.com/pulse/true-cost-hosting-your-own-llm-comprehensive-comparison-binoloop-l3rtc
29. https://www.ibm.com/think/topics/llm-temperature
30. https://lukesalamone.github.io/posts/what-is-temperature/
31. https://gptforwork.com/guides/openai-gpt3-temperature
32. https://www.techtarget.com/searchenterpriseai/tip/Understanding-the-role-of-temperature-settings-in-AI-output
33. https://www.projectpro.io/article/llm-temperature/1073
34. https://www.linkedin.com/pulse/creatively-deterministic-what-temperature-topp-ai-kevin-tupper
35. https://www.vellum.ai/llm-parameters/temperature
36. https://dagshub.com/glossary/llm-temperature/
37. https://www.metriccoders.com/post/what-is-the-temperature-parameter-in-language-models-and-how-to-set-it
38. https://www.linkedin.com/pulse/mastering-text-generation-unveiling-secrets-decoding-strategies-jain-rqwtf
39. https://www.assemblyai.com/blog/decoding-strategies-how-llms-choose-the-next-word/
40. https://www.scaler.com/topics/nlp/decoding-strategies-for-transformers/
41. https://www.scaler.com/topics/nlp/decoding-methods/
42. https://www.packtpub.com/en-us/learning/how-to-tutorials/exploring-token-generation-strategies
43. https://www.ibm.com/docs/en/watsonx/saas?topic=lab-model-parameters-prompting
44. https://aman.ai/primers/ai/token-sampling/
45. https://www.pingcap.com/article/decoding-methods-compared-top-k-and-other-token-selection-techniques/
46. https://bookdown.org/tranhungydhcm/mybook/basics-of-large-language-models.html
47. https://www.metriccoders.com/post/defining-stopping-criteria-in-large-language-models-a-practical-guide
48. https://discuss.huggingface.co/t/implimentation-of-stopping-criteria-list/20040
49. https://stackoverflow.com/questions/77549942/stopping-criteria-for-llama-2-does-not-work
50. https://flyte.org/blog/getting-started-with-large-language-models-key-things-to-know
51. https://huggingface.co/blog/constrained-beam-search
52. https://www.vellum.ai/llm-parameters/stop-sequence
53. https://attri.ai/generative-ai-wiki/llm-optimization-parameters
54. https://www.reddit.com/r/PromptEngineering/comments/14jzpez/what_are_the_practical_applications_of_stop/
55. https://help.promptitude.io/en/articles/8897040-stop-sequence-understanding-setting-it-correctly
56. https://developer.nvidia.com/blog/how-to-get-better-outputs-from-your-large-language-model/
57. https://community.openai.com/t/how-to-add-stop-sequence-for-the-model-gpt-3-5-turbo-1106-fine-tuning/585741

58. https://learnprompting.org/blog/llm-parameters
59. https://help.openai.com/en/articles/5072263-how-do-i-use-stop-sequences-in-the-openai-api
60. https://www.youtube.com/watch?v=G7B8Smt_dx4
61. https://adasci.org/in-context-learning-vs-rag-in-llms-a-comprehensive-analysis/
62. https://www.prompthub.us/blog/in-context-learning-guide
63. https://www.lakera.ai/blog/what-is-in-context-learning
64. https://floatbot.ai/tech/In-context-learning-llms
65. https://research.ibm.com/blog/demystifying-in-context-learning-in-large-language-model
66. https://www.restack.io/p/generative-ai-answer-in-context-learning-cat-ai
67. https://www.hopsworks.ai/dictionary/in-context-learning-icl
68. https://www.reddit.com/r/aipromptprogramming/comments/1g8uimj/prompt_engineering_best_practices_for_incontext/
69. https://ai.stanford.edu/blog/understanding-incontext/
70. https://news.mit.edu/2023/large-language-models-in-context-learning-0207
71. https://prompthub.substack.com/p/using-llms-to-generate-in-context
72. https://github.com/EgoAlpha/prompt-in-context-learning/blob/main/PromptEngineering.md
73. https://www.datacamp.com/tutorial/few-shot-prompting
74. https://symbio6.nl/en/blog/what-is-few-shot-prompting
75. https://www.prompthub.us/blog/the-few-shot-prompting-guide
76. https://learnprompting.org/docs/basics/few_shot
77. https://www.promptingguide.ai/techniques/fewshot
78. https://www.ibm.com/think/topics/ai-hallucinations
79. https://www.techtarget.com/whatis/definition/AI-hallucination
80. https://www.metriccoders.com/post/what-is-hallucination-and-how-can-it-be-controlled-using-prompt-engineering
81. https://nexla.com/ai-infrastructure/llm-hallucination/
82. https://kata.ai/blog/prompt-engineering-method-to-reduce-ai-hallucinations/
83. https://milvus.io/ai-quick-reference/how-can-prompt-engineering-help-mitigate-hallucinations-eg-telling-the-llm-if-the-information-is-not-in-the-provided-text-say-you-dont-know
84. https://documentation.suse.com/suse-ai/1.0/html/AI-preventing-hallucinations/index.html
85. https://www.prompthub.us/blog/three-prompt-engineering-methods-to-reduce-hallucinations
86. https://symbio6.nl/en/blog/prompting-strategies-prevent-ai-hallucinations
87. https://www.machinelearningmastery.com/a-gentle-introduction-to-hallucinations-in-large-language-models/
88. https://www.metriccoders.com/post/how-to-improve-llm-reasoning-when-your-chain-of-thought-cot-prompt-fails
89. https://blog.gopenai.com/unlocking-reasoning-power-in-llms-how-cot-prompting-revolutionizes-problem-solving-58233bf3c218
90. https://www.metriccoders.com/post/how-to-improve-llm-reasoning-if-your-chain-of-thought-cot-prompt-fails

91. https://cameronrwolfe.substack.com/p/chain-of-thought-prompting-for-llms
92. https://www.youtube.com/watch?v=akj6v0Ykh1Y
93. https://arxiv.org/html/2402.10200v2
94. https://openreview.net/forum?id=4Zt7S0B0Jp