

# Prompting, Reasoning, & Memory

Shreya Havaladar

---

# Today's Lecture

1. **Prompting**
2. Reasoning techniques
3. Memory management

# Prompts!

Key framing:

**Prompts are not questions; they are specifications**

# Prompts as specifications

## **Main components:**

1. Goal (what success means)
2. Inputs (what is provided; what is missing)
3. Constraints (must/never; policy; style; length)
4. Output format (schema; sections; deterministic formatting)
5. Acceptance tests (2–5 examples of correct/incorrect outputs)

# Prompt Engineering

Prompts always involve instructions and context passed to an LLM to achieve a desired task.

**Prompt engineering** is the practice of developing and optimizing these prompts.

# Typical prompt layout

Use consistent blocks to reduce ambiguity and improve debugging:

**Role/Task:** “You are ... your task is ...”

**Rules:** bullet list, straightforward information

**Context:** any needed facts or data

**Output format:** exact schema + “no extra text”

**Examples:** correct and incorrect (if needed)

# Design Tips

1. Start simple
  - a. Prompting is an iterative process!
  - b. Experiment, tweak the prompt, repeat

Three main things to remember:

**specificity, simplicity, and conciseness**

# Design Tips

2. Break things down into subtasks
  - a. LLMs are (usually) bad at doing many complex things at once
  - b. Helps you keep your prompts simple and specific.
  - c. Iteratively break down components your LLM struggles with

# Design Tips

3. Be clear about the task and expected output
  - a. The more descriptive the prompt is, the better the results
  - b. But... too many (unnecessary) details may degrade results
  - c. Examples are almost always beneficial

# A bad prompt

“Explain the concept prompt engineering. Keep the explanation short, only a few sentences, and don't be too descriptive”

- Length of explanation ❌
- Style of explanation ❌

# A better prompt

“Use 2-3 sentences to explain the concept of prompt engineering to a high school student.”

- Length of explanation 
- Style of explanation 

# A bad prompt

“You are an agent that recommends movies to a customer. DO NOT ASK FOR INTERESTS. DO NOT ASK FOR PERSONAL INFORMATION.”

- Output constraints ✘
- Edge cases ✘

# A better prompt

“You are an agent that recommends movies to a customer. The agent should recommend a movie from the top global trending movies. It should not ask users for their preferences or personal information. If the agent doesn't have a movie to recommend, it should respond “Sorry, no movie available””

- Output constraints 
- Edge cases 

# Prompting templates

Templates exist for most basic LLM tasks ([like those here!](#))

- Text Summarization
- Information Extraction
- Question Answering
- Text Classification
- Code Generation

Also useful: <https://chatgpt.com/use-cases/students>

# Activity (15 min)

Image generation models typically require prompts that strike a balance between simplicity and specificity.

Go to [guessprompt.com](https://guessprompt.com)

**Prize of 1 late day for whoever gets the highest score on the leaderboard!**

*As always, AI tools are allowed*



# Today's Lecture

1. Prompting
2. **Reasoning techniques**
3. Memory management

# What is reasoning?

Reasoning in LLMs refers to the capability of models to solve complex, multi-step problems by breaking them down into intermediate, logical, and sequential steps

# What is reasoning?

Reasoning in LLMs refers to the capability of models to solve complex, multi-step problems by breaking them down into intermediate, logical, and sequential steps

**Reasoning in prompts:** Taking any LLM and prompting it in a way that elicits reasoning capabilities

**Reasoning models:** LLMs that have been fine-tuned to internally reason before generating an output

# When reasoning prompts help

## **Reasoning techniques typically help for:**

- Multi-step arithmetic
- Multi-constraint decisions
- Compositional tasks
- Multi-hop QA.

## **They often don't help for:**

- Missing knowledge
- Ambiguous tasks
- Tasks better solved by tools

# Chain-of-Thought (CoT)

CoT improves multi-step reasoning by eliciting intermediate steps  
→ “let’s think step-by-step”

---

## Chain-of-Thought Prompting Elicits Reasoning in Large Language Models

---

Jason Wei   Xuezhi Wang   Dale Schuurmans   Maarten Bosma  
Brian Ichter   Fei Xia   Ed H. Chi   Quoc V. Le   Denny Zhou

Google Research, Brain Team  
{jasonwei, dennyzhou}@google.com

# Chain-of-Thought (CoT)

Types of CoT prompting:

**Zero-shot:** *task description only, with no examples given*

**Few-shot:** *task description with some amount of examples*

**8-shot:** *task description with exactly 8 examples*

When to use CoT: multi-step math, symbolic manipulation, compositional reasoning, planning subproblems.

# Decomposition

Generic scaffold:

List the constraints

Break into 3–6 substeps

Solve each substep

Assemble final answer in the required format

“Least-to-most” style decomposition: Start with easy subquestions; use them as givens for the harder step.

# Self-consistency

Uses sampling to choose the highest likelihood answer

1. Set the temperature to a higher value
  - a. → increases diversity in generations
2. Have the LLM produce N independent solutions
3. Choose the majority, or “most consistent” generation

# Tree of Thoughts (ToT)

Generic scaffold:

```
Define "thought" unit (partial solution / subplan)
Generate k candidates
Score candidates (model-based evaluation, verifier)
Expand best nodes until depth/timeout
Return best leaf + justification
```

Generalizes CoT into a search over "thoughts" (branches), enabling exploration and lookahead, improving problems requiring search.

# Acting + tools

## **ReAct (reason + act):**

ReAct interleaves reasoning traces with actions, addressing hallucination by fetching external information during reasoning.

## **PAL (program-aided language):**

PAL offloads execution correctness to a runtime (e.g., Python), reducing arithmetic/logical slip-ups in final computation.

# Reflexion

Reflexion converts feedback into self-reflections stored in memory.

These reflections are re-used in subsequent trials.

→ Improving performance without fine-tuning.

Generic scaffold:

Attempt → Evaluate → Write reflection → Next attempt  
conditioned on reflection

# Self-verification

The LLM has to “verify” it’s answer before continuing

- **Self-consistency:** Majority vote for single-answer tasks.
- **Checker model:** critique + conditioned regeneration.
- **Tool-based verification:** compute with interpreter; retrieve citations; cross-check constraints.
- **Invariant checks:** schema validity (e.g. correct json), policy constraints, numeric sanity checks

# Reasoning failure modes

- **Constraint drop:** the LLM ignores a must/never rule
- **Premature closure:** answers before considering all constraints
- **Spurious precision:** invents missing values
- **Non-robust in long context:** fails when the relevant fact is not near the beginning/end

*\* Research shows that LLM performance can degrade when relevant information is positioned in the middle of a long input prompt, even for long-context models.*

# Reasoning Models

Prompt engineering for complex reasoning problems is a challenging task. This motivated reasoning models!

**Reasoning models generate internal chains of intermediate steps, then select and refine a final answer.**

# Recap: standard LLM inference

During generation, an LLM:

1. Tokenizes the input
2. Runs a forward pass
3. Gets a set of logits for every token in its vocabulary
4. Converts these logits to a probability distribution over tokens
5. Selects the output token using this distribution
6. Appends the output token to the generation; repeats

# Reasoning LLM inference

Same as standard inference, but...

- The LLM first generates **“thinking tokens”**
  - Structured intermediate reasoning traces
  - Thinking step-by-step, decomposition, verification, etc.
- These “thinking tokens” are not part of the final answer

ChatGPT o3 ▾

↑ Share ...

Can you design a lecture on safety mechanisms for Agentic AI?

Thought for 4 seconds >

**Lecture Title:** Safety Mechanisms for Agentic AI

**Audience & Length:** Graduate CS/NLP students, 90 minutes

## 1 Learning Objectives

- Recognize risk vectors unique to autonomous agents.
- Distinguish technical vs. governance safety layers.
- Evaluate and design concrete safeguards for agent deployments.

## 2 Agenda

Time	Topic	Key Points	Method
------	-------	------------	--------

Activity · 0s

×

## Thinking

- **Designing a lecture on safety mechanisms for Agentic AI**  
The user is asking for a lecture design, which includes an outline, objectives, topics, reading, discussion points, case studies, and evaluation. They're likely aiming for a concise, formal structure—graduate-level, around 90 minutes. No need for a slides deck or document unless they want one later. I'll focus on a structured outline.
- **Designing a concise lecture on Agentic AI safety mechanisms**  
The user wants a formal, concise lecture design on safety mechanisms for Agentic AI. I don't need to browse, as the concept is stable and citations are optional. I'll focus on creating a structured, bullet-point outline: introduction, core safety principles, and more.

# Application: jailbreaking

Jailbreaking an LLM is the process of crafting specific, often deceptive, input prompts designed to bypass the built-in safety, ethical, and operational restrictions of an AI model.

**Jailbreaking game: [gandalf.lakera.ai](https://gandalf.lakera.ai)**

# Today's Lecture

1. Prompting
2. Reasoning techniques
3. **Memory management**

# Memory

**“Memory” for an LLM means context selection and presentation.**

- Your LLM should “remember” pieces of information from prior queries
- You must strategically manage relevant information from prior exchanges to maintain coherence.

# Sequential chain

**In sequential memory chaining, you append everything.**

Pros: Easy to implement, you append each new turn and include the entire conversation chain as the input

Cons: Quickly runs into context limits. Only useful for very short sessions

# Sliding window

**In sliding window memory, you append only the N last turns.**

Pros: Also easy to implement, you keep a fixed-size recent window and drop older turns.

Cons: Does not run into context limits, but can drop important information from earlier turns

# Summary-based memory

**In summary-based memory, you periodically summarize the conversation**

Pros: You can maintain longer-term context under token limits

Cons: Requires iteration to determine how to best summarize information for your specific application

# Activity (20 min, groups of 3-4)

**Step 1:** Think of an LLM task that requires multiple turns, has evolving constraints, depends on remembering prior decisions, and fails if earlier information is forgotten

**Step 2:** Design a memory store that includes (1) a method to extract relevant information from a conversation, and (2) a storage schema for this information ***under 500 total tokens***