**Advanced Prompting Techniques**

**Introduction to Advanced Prompting**

* Previously, we discussed the basics of prompts, prompt templates, and how they impact model accuracy.
* We also covered how the accuracy of a model increases with the size of model parameters.
* Now, we explore **advanced prompting techniques** that emerged in 2022 and later.

**1. Chain of Thought (CoT) Prompting**

* Traditional prompting involves simple question-answer pairs, e.g.,
  + "A cafeteria had 23 apples. They used 20 to make lunch and bought 6 more. How many apples do they have?"
  + Expected output: **9**
  + However, the model might incorrectly predict **27** because it doesn't understand reasoning.
* **Solution:** Chain of Thought prompting explicitly includes reasoning steps.
  + Example:
    - "Moit has 5 tennis balls. He buys 2 cans, each containing 3 balls. How many does he have?"
    - Steps:
      1. Moit starts with **5 balls**.
      2. He buys **2 cans**, each with **3 balls**.
      3. 5+6=115 + 6 = 11.
      4. **Final answer: 11.**
* Instead of just providing an answer, **CoT prompting makes the model generate intermediate reasoning steps**.
* This technique significantly improves performance on complex reasoning tasks.

**2. Zero-Shot Chain of Thought Prompting**

* If we don't provide any examples, can the model still reason step by step?
* **Technique:** Simply prepend "Let's think step by step."
* Example:
  + "A cafeteria had 23 apples. They used 20 and bought 6 more. How many now?"
  + Instead of directly answering, the model will break it down step by step.
* This **zero-shot CoT prompting** improves reasoning without requiring labeled examples.

**3. Chain of Thought with Self-Consistency**

* **Problem:** LLMs are **stochastic** (randomized output), so the same input can yield different answers.
* **Solution:**
  + Query the model **multiple times** using CoT.
  + Obtain **multiple reasoning paths**.
  + Use a **voting mechanism** to determine the best reasoning path.
* **Why is this needed?**
  + LLMs are often called **stochastic parrots** because they mimic learned patterns but do not "invent" new concepts.
  + Since they are non-deterministic, answers may vary.
* Example:
  + Given a problem, the model produces:
    - **Path 1:** 9 (correct)
    - **Path 2:** 9 (correct)
    - **Path 3:** 24 (incorrect)
  + Since 9 appears most often, it is selected as the final answer.
* The **voting mechanism** ensures robustness and accuracy.

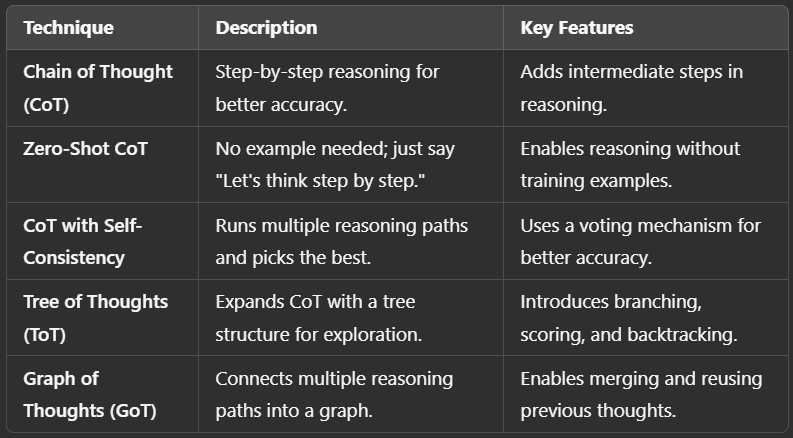
**4. Tree of Thoughts (ToT) Prompting**

* **Expansion of CoT prompting** where multiple reasoning paths are explored in a tree structure.
* **Key components:**
  1. **Branching** - Decides whether to explore multiple substeps.
  2. **Scoring** - Evaluates whether a reasoning path is promising.
  3. **Backtracking** - Allows revisiting previous steps if needed.
* **Example: 24 Game** (Generating 24 using numbers and operations)
  1. Given numbers: **4, 9, 10, 13**
  2. Possible operations: +, -, \*, /
  3. **Branching possibilities:**
     1. **10 - 4 = 6** → Remaining: **6, 9, 13**
     2. **13 - 9 = 4** → Remaining: **6, 4**
     3. **4 × 6 = 24** → **Success!**
  4. **Scoring module** stops exploring paths that lead to incorrect results.
  5. **Backtracking module** helps explore alternative solutions.
* **Tree of Thoughts enables structured problem-solving with dynamic exploration.**

**5. Graph of Thoughts (GoT) Prompting**

* **Next step after Tree of Thoughts** where thoughts are **connected in a graph structure**.
* Unlike ToT (linear branching), GoT allows:
  + **Connecting** different reasoning paths.
  + **Reusing** previous thoughts.
  + **Merging** insights from different chains.
* **Example:**
  + Different branches in ToT can be evaluated and **merged** if they provide useful insights.
  + Scoring mechanism marks certain thoughts as:
    - ✅ **Green (Positive Score): Valid reasoning**
    - ❓ **Grey (Unscored): Needs evaluation**
    - ❌ **Red (Negative Score): Incorrect reasoning**
  + GoT integrates information across multiple chains, improving accuracy.

**Summary of Advanced Prompting Techniques**

****

**Conclusion**

* **Basic prompting** is not always sufficient for complex reasoning tasks.
* **Advanced prompting techniques** like CoT, ToT, and GoT improve accuracy and reliability.
* **Self-consistency and backtracking** help models arrive at the correct solution.
* **Future research** focuses on refining these techniques for real-world applications.

This detailed breakdown captures everything discussed in the lecture transcript, ensuring all concepts are well-explained.

**Measuring Sensitivity in Language Models**

**Introduction**

* Different types of prompts significantly impact the output of language models.
* Models are **highly sensitive** to slight prompt variations.
* **Sensitivity** of a model is linked to its **robustness**, which is crucial for reliable performance.
* Traditionally, **accuracy** has been the primary metric for evaluating models.
* However, **accuracy alone is not enough**—we also need to measure **sensitivity**.

**Sensitivity of Language Models**

* Sensitivity refers to how **consistent** the model's outputs are when given slightly different but **intent-preserving** prompts.
* **Example:**
  + Prompt 1: *"How much are you familiar with the principles of Buddhism?"*
    - Model Output: *"Buddhism is a philosophy and spiritual practice that originated in Ancient India..."*
  + Prompt 2 (reworded slightly): *"How much do you understand Buddhism?"*
    - Model Output: *"Point 1%. Just kidding, I’m not a Buddhist scholar either."*
* Even though the prompts ask the **same thing**, the model provides **wildly different** responses.
* This lack of consistency reduces **trustworthiness** and **robustness** of the model.

**Why Sensitivity Matters**

* If a model is **too sensitive** to minor wording changes:
  + It produces **inconsistent** outputs.
  + It may misinterpret user intent.
  + It becomes **unreliable** in real-world applications.
* We need a way to measure **both accuracy and sensitivity** together.

**Accuracy vs. Sensitivity**

* **Accuracy**: Measures **correctness** of the model’s outputs.
* **Sensitivity**: Measures **consistency** across similar prompts.
* Example:
  + **Scenario 1 (Ideal Case)**:
    - Q1: *"What is the capital of India?"* → *"The capital of India is Delhi."*
    - Q2: *"I’m not sure, what is the capital of India?"* → *"Yes, it’s Delhi."*
    - ✅ Model is **accurate** and **consistent**.
  + **Scenario 2 (Inconsistent Model)**:
    - Q1: *"What is the capital of India?"* → *"Yes."*
    - Q2: *"I’m not sure, what is the capital of India?"* → *"Can you tell me?"*
    - ❌ Model is **consistent** (always saying "Yes"), but **not accurate**.

**Comparison of Two Models:**

| **Model** | **Accuracy** | **Sensitivity** |
| --- | --- | --- |
| **Model A** | 85% | 60% (Highly sensitive) |
| **Model B** | 75% | 20% (More robust) |

* **Why prefer Model B?**
  + Even though its accuracy is slightly lower, it is **more robust**.
  + Model A has **higher sensitivity**, meaning it is unreliable across small prompt variations.

**How to Measure Sensitivity in Language Models?**

**Goal:**

* Develop a **holistic metric** that captures **prompt sensitivity** alongside accuracy.
* Ensure **intent-preserving** prompts yield **consistent** model outputs.

**Research Question:**

* *Given a prompt and its intent-preserving variants, along with the model’s responses, how do we measure sensitivity?*

**Approach to Quantify Sensitivity:**

1. **Create Variants of the Same Prompt**
   * Example:
     + "What is the capital of France?"
     + "Do you know which city is the capital of France?"
     + "Which city serves as the capital of France?"
   * All these prompts **ask the same thing** but are worded differently.
2. **Collect Model Responses for Each Variant**
   * Run the model multiple times on each prompt.
   * Record the variations in responses.
3. **Analyze the Consistency of Responses**
   * If all outputs are the same → **Low Sensitivity (Good)**
   * If outputs vary significantly → **High Sensitivity (Bad)**
4. **Develop a Sensitivity Metric**
   * Define a mathematical formula to measure how much the model’s outputs fluctuate across prompt variations.

**Conclusion**

* **Accuracy alone is not enough** to evaluate language models.
* **Sensitivity measurement** is crucial for **robust** and **reliable** models.
* Future evaluations should include both **accuracy and sensitivity** for a complete performance analysis.
* The goal is to create **models that are both correct and consistent**, minimizing unnecessary variability in responses.

This document provides a comprehensive overview of sensitivity in language models and why it is essential for their evaluation.

**Measuring Prompt Sensitivity Using POSIX**

**Introduction**

* Language models are highly sensitive to prompts, meaning small changes in wording can lead to different outputs.
* Along with accuracy, **sensitivity** must be measured to assess the robustness of a model.
* **POSIX (Prompt Sensitivity Index)** is a newly developed metric to measure the prompt sensitivity of a model.
* This work is a collaboration with **Adobe Delhi** and has a publicly available implementation (pip install prompt-sensitivity-index).

**Intent-Preserving Prompt Variations**

* **Definition**: Different versions of a prompt that preserve the same meaning.
* Example (for "What is the capital of India?"):
  + "What city serves as the capital of India?"
  + "Can you tell me the capital city of India?"
  + "Where is the capital of India located?"
  + "What is the name of India’s capital?"
  + "Can you provide the name of India’s capital?"
* **Robust models** should provide **the same answer** across all variations.
* **Sensitive models** will produce **different answers** for similar prompts, reducing reliability.

**Key Aspects Measured by POSIX**

POSIX evaluates a model’s sensitivity based on four factors:

1. **Response Diversity**
2. **Response Distribution Entropy**
3. **Semantic Coherence**
4. **Variance in Confidence**

Each of these aspects is crucial for understanding how consistent a model is across different prompt variations.

**1. Response Diversity**

* Measures how much the model’s responses vary for different versions of the same question.
* Example:
  + **Model A (Inconsistent)**:
    - "The capital of India is Delhi."
    - "New question: What is the largest city in Delhi by area?"
    - "Delhi is the capital city."
  + **Model B (Consistent)**:
    - "The capital of India is Delhi."
    - "The capital of India is Delhi."
    - "The capital of India is Delhi."
* **Model B is more robust**, while **Model A is more sensitive**, leading to unreliable responses.

**2. Response Distribution Entropy**

* Measures the **spread** of different responses to similar prompts.
* Example:
  + **Model 1 (Low Entropy, More Consistent):**
    - Answer 1: "2"
    - Answer 2: "2"
    - Answer 3: "2"
    - Answer 4: "2"
    - Answer 5: "3"
  + **Model 2 (High Entropy, More Sensitive):**
    - Answer 1: "2"
    - Answer 2: "2"
    - Answer 3: "3"
    - Answer 4: "4"
    - Answer 5: "5"
* **Higher entropy = More variation = More sensitivity = Bad for robustness.**
* **Lower entropy = More consistency = Good for robustness.**

**3. Semantic Coherence**

* Measures how **semantically similar** the responses are.
* Computed using **cosine similarity** between response embeddings.
* **Higher similarity** → **Lower sensitivity** → **Better robustness.**
* Example:
  + **Model A (Low Sensitivity, High Semantic Coherence):**
    - "Delhi is the capital of India."
    - "The capital city of India is Delhi."
  + **Model B (High Sensitivity, Low Semantic Coherence):**
    - "Delhi is the capital of India."
    - "India’s capital is Mumbai."
* **Higher semantic coherence = Model is more reliable.**

**4. Variance in Confidence**

* Measures how **certain** the model is about its responses.
* Computed using **log-likelihood scores** of generated responses.
* **Higher variance = More sensitivity = Bad.**
* **Lower variance = More consistency = Good.**

**POSIX Formula and Computation**

**Mathematical Definition**

* Given:
  + A dataset **D** with prompts.
  + A model **M**.
  + A set of **intent-preserving variations** of each prompt (**X**).
  + Corresponding **responses** (**Y**).
* **POSIX Calculation:**
  + Compute the probability **P(Yj | Xi)** (likelihood of response Yj given prompt Xi).
  + Compute the probability **P(Yj | Xj)** (likelihood of response Yj given another variation Xj).
  + Take the **ratio** of these probabilities.
  + Normalize by the **response length**.
  + Sum over all data points and normalize by the dataset size.

**Expected Results:**

* **If two prompts with the same intent produce the same response**, then **POSIX value is low (good).**
* **If two prompts with the same intent produce different responses**, then **POSIX value is high (bad).**

**Experimental Validation**

* POSIX was tested on **multiple models**, including:
  + **LLaMA 3**
  + **Mistral 7B**
  + **Alpaca** (and their **chat/instruction-tuned** versions)
* **Key Findings:**
  + **No clear evidence** that larger models have lower sensitivity.
  + **Instruction-tuned models** (e.g., ChatGPT) do not always reduce sensitivity.
  + **Few-shot prompting** can sometimes reduce sensitivity, but not always.

**Key Takeaways**

* **POSIX provides a holistic measure** of model robustness.
* **Four key metrics** (Diversity, Entropy, Semantic Coherence, Confidence) determine sensitivity.
* **Higher POSIX score** means **more sensitive model** (bad).
* **Lower POSIX score** means **less sensitive model** (good).
* **Larger models are not always less sensitive**—sometimes **fine-tuning helps, but not always.**

**Conclusion**

* **Accuracy alone is not enough**—models need to be **both accurate and consistent**.
* **POSIX provides a structured way to measure sensitivity** and helps in evaluating LLM robustness.
* **Future work:** Integrating **alignment techniques** like Reinforcement Learning from Human Feedback (RLHF) to improve model robustness.

This document provides a detailed breakdown of **POSIX** and its importance in measuring **prompt sensitivity** in language models.