

Generative and Agentic AI in Practice

DS 246 (1:2)

Generative and Agentic AI in Practice: DS 246 (1:2)

Prof. Sashikumaar Ganesan

Lecture Topics

Phase 1: Foundations (Weeks 1-8, leading to Midterm)

- Week 2 (Aug 13): Generative AI Core Concepts & Architecture
 - Topics: Prompting Principles, Patterns, Techniques (Zero-Shot, CoT, ReAct), Prompt Chaining, LLM settings
- **Assignment**
 - Achieve a target output using different prompting techniques and compare performance

Prompting Principles

Prompting Principle



Working Definition

A methodological approach to design effective input text (prompts) that directly influences how LLMs understand, process, and respond to user prompts with accurate and contextually appropriate outputs.

Poor Prompting

Prompt: "Who won the 2023 Nobel Prize in Computing?"

LLM Response: The 2023 Nobel Prize in Computing was awarded to Dr. Sarah Chen for her groundbreaking work in quantum algorithm optimization, making her the first recipient of this prestigious award.

Problem: This is a complete hallucination - there is no Nobel Prize in Computing. The LLM generated a confident, detailed, but entirely fictional response, even creating a fake person and achievement.

Correct Prompting

Prompt: "Based on the following information, answer the question accurately. If the information is insufficient, say 'Cannot determine from provided information.'" Context: The Nobel Foundation awards prizes in Physics, Chemistry, Physiology or Medicine, Literature, Peace, and Economic Sciences. There is no Nobel Prize for Computing. The closest related award is the Turing Award, often called the "Nobel Prize of Computing." Question: Who won the 2023 Nobel Prize in Computing?

Why this works : The prompt follows good principles by:

- Providing relevant context (external information)
- Setting clear instructions for handling insufficient information
- Establishing boundaries for factual accuracy
- Reducing the chance of hallucination through

Five-Level Framework for Working with LLMs



1. Basic Prompting (Conversational Level)

Users interact with LLMs through chatbot interfaces (ChatGPT, Claude, etc.) by writing natural language prompts.

Characteristics

- No coding required
- User crafts effective prompts through trial and error
- Limited control over model behavior
- Example: "Explain quantum computing in simple terms"

Limitations

- Inconsistent results, no systematic optimization, limited reproducibility

Five-Level Framework for Working with LLMs



2. Prompt Engineering (API Level)

Systematic design of prompt templates with API calls, controlling parameters like temperature and top_p for consistent, optimized outputs.

Key Parameters

- Temperature (0.0-2.0): Controls randomness/creativity
- Top_p (0.0-1.0): Nucleus sampling for output diversity
- Max_tokens: Output length control
- System prompts: Role and behavior definition

Advantages

- Advantages: Reproducible, programmable, batch processing capable

```
python

response = api.call(
    prompt="Translate: {text}",
    temperature=0.3, # Low for consistency
    top_p=0.9,
    max_tokens=100
)
```

Five-Level Framework for Working with LLMs



3. RAG (Retrieval-Augmented Generation)

Enhances LLM responses by retrieving relevant information from external knowledge bases and augmenting the prompt with contextual data.

Process

- Retrieve relevant documents from database
- Augment prompt with retrieved context
- Generate response grounded in facts

Benefits

- Reduces hallucinations
- Provides up-to-date information
- Enables domain-specific responses
- Source citation capability

Use Cases

- Customer support, documentation Q&A, knowledge management systems

Five-Level Framework for Working with LLMs



4. Fine-Tuning (Parameter-Efficient Methods)

Adapting pre-trained models to specific tasks by training additional parameters while keeping the base model frozen.

Popular Methods

- LoRA (Low-Rank Adaptation): Adds trainable low-rank matrices ($\sim 0.1-1\%$ of original parameters)
- QLoRA: Quantized version using 4-bit precision for memory efficiency
- DoRA: Weight-decomposed adaptation for better performance
- Adapters: Small trainable modules inserted between layers

Advantages

- Reduces hallucinations
- Provides up-to-date information
- Enables domain-specific responses
- Source citation capability

Example Applications

- Domain-specific chatbots, specialized translation, custom writing styles

Five-Level Framework for Working with LLMs



5. Training LLMs from Scratch

Building and training a new language model from random initialization on custom datasets.

Requirements

- Massive datasets (terabytes of text)
- Significant computational resources (100s-1000s of GPUs)
- Months of training time
- Millions of Rupees in costs

When to Consider

- Highly specialized domains with unique vocabulary
- Proprietary architectures needed
- Complete control over model behavior required
- Security/privacy constraints prohibiting external models

Five-Level Framework for Working with LLMs



Decision Framework

- **Prompting:** Quick prototypes, one-off tasks
- **Prompt Engineering:** Production applications, consistent outputs needed
- **RAG:** When current information or domain knowledge is critical
- **Fine-tuning:** Task-specific performance requirements, resource constraints
- **Training from Scratch:** Complete control needed, unlimited resources

Cost-Benefit Analysis

- Prompting: ₹0 setup, variable quality
- Prompt Engineering: Low cost, good control
- RAG: Moderate setup cost, high accuracy
- Fine-tuning: ₹100K-1000K, specialized performance
- Training: ₹1M+, complete customization

The Anatomy of an Effective Prompt

NOTE: Most of the Chatbots, incorporated these techniques, and we need it only when we built our own Chatbots using API calls to LLMs

The Anatomy of an Effective Prompt

Key Element 1: Structure

The structure of a prompt refers to how it is organized to guide the AI's understanding and response generation.

- **Correct:**
 - **Clear and concise instructions:** The prompt should provide a well-defined starting point, outlining the task and desired outcome.
 - **Logical flow:** Structure the prompt in a way that guides the AI step-by-step, ensuring a coherent thought process.
- **Incorrect:**
 - **Vague or open-ended instructions:** Without clear direction, the AI may generate irrelevant or nonsensical outputs.
 - **Disorganized structure:** A jumbled prompt can confuse the AI, leading to unpredictable results.

The Anatomy of an Effective Prompt

Key Element 1: Structure : Examples

- Ex1:
 - Incorrect: "Write something on global warming."
 - Correct: "Write an informative article on the causes and effects of global warming."
 - *Explanation: The correct prompt provides a clear structure directing the AI to produce an article format with specified content.*
- Ex2:
 - Incorrect: "Make a blog."
 - Correct: "Create a blog post about recent advancements in AI technology."
 - *Explanation: The correct prompt specifies the type of content, making it clear that a detailed blog post is required.*

The Anatomy of an Effective Prompt

Key Element 2: Clarity

The language used in the prompt should be unambiguous and easy for the AI to comprehend.

- **Correct:**
 - **Simple and direct language:** Avoid complex jargon or overly technical terms.
 - **Precise vocabulary:** Use words that have clear and defined meanings in the context of the task.
- **Incorrect:**
 - **Ambiguous or figurative language:** Language open to interpretation can lead to misconstrued prompts and unexpected outputs.
 - **Informal or conversational language:** Maintain a professional tone while keeping the language clear and concise.

The Anatomy of an Effective Prompt

Key Element 2: Clarity: Examples

- Ex1:
 - Incorrect: "Do the needful for the report."
 - Correct: "Please update the final report with the latest sales figures and conclusions."
 - *Explanation: The correct prompt clearly communicates the specific actions needed.*
- Ex2:
 - Incorrect: "Book info."
 - Correct: "Provide a summary of the book titled 'Brave New World' by Aldous Huxley."
 - *Explanation: The correct prompt explicitly states what information is needed and about which book.*

The Anatomy of an Effective Prompt

Key Element 3: Precision

Precision in the prompt is crucial for achieving the desired level of detail and accuracy in the AI's response. It involves using specific terms and details that tailor the AI's output to your exact needs.

- **Correct:**
 - **Specific details and instructions:** The more information you provide, the better the AI can tailor its response to your needs.
 - **Examples and references (when applicable):** Including relevant examples or references can further refine the AI's understanding of your intent.
- **Incorrect:**
 - **Generic or lacking details:** A vague prompt will likely result in a generic or uninformative response from the AI.
 - **Focus on quantity over quality:** Don't overload the prompt with unnecessary details that might distract the AI from the core objective.

The Anatomy of an Effective Prompt

Key Element 3: Precision : Examples

- Ex1:
 - Incorrect: "Write about technology."
 - Correct: "Write a detailed comparison between quantum computing and classical computing."
 - *Explanation: The correct prompt narrows down the topic to a specific comparative analysis.*
- Ex2:
 - Incorrect: "Describe dog."
 - Correct: "Describe the physical characteristics and temperament of a German Shepherd."
 - *Explanation: The correct prompt is specific about the breed and aspects to be described.*

Core Prompting Techniques

The Anatomy of an Effective Prompt

Key Element 3: Precision : Examples

- Ex1:
 - Incorrect: "Write about technology."
 - Correct: "Write a detailed comparison between quantum computing and classical computing."
 - *Explanation: The correct prompt narrows down the topic to a specific comparative analysis.*
- Ex2:
 - Incorrect: "Describe dog."
 - Correct: "Describe the physical characteristics and temperament of a German Shepherd."
 - *Explanation: The correct prompt is specific about the breed and aspects to be described.*

Basic Chain-of-Thought (CoT) Prompting

Step-by-Step Reasoning

Without CoT:

- Prompt: "What is 28×47 ?"
- Response: "1316"
- Limitation: Direct answer without showing work, making it difficult to verify or learn from the process.

With CoT:

- Prompt: "What is 28×47 ? Show your step-by-step calculation."
- Response: "Let me calculate 28×47 step by step:
 - $28 \times 40 = 1120$
 - $28 \times 7 = 196$
 - $1120 + 196 = 1316$
 - Therefore, $28 \times 47 = 1316$ "
- Benefit: Each step is visible and verifiable, allowing for error detection and learning.

Basic Chain-of-Thought (CoT) Prompting

Breaking Down Complex Problems

Without CoT:

- Prompt: "Is this investment profitable if it costs \$10,000 with 15% annual return over 3 years?"
- Response: "Yes, it's profitable."
- Limitation: No insight into the calculation or assumptions made.

With CoT:

- Prompt: "Is this investment profitable if it costs \$10,000 with 15% annual return over 3 years? Walk through your analysis."
- Response: "Let me break this down:
 - Initial investment: \$10,000 Year 1: $\$10,000 \times 1.15 = \$11,500$ Year 2: $\$11,500 \times 1.15 = \$13,225$ Year 3: $\$13,225 \times 1.15 = \$15,208.75$
 - Total profit: $\$15,208.75 - \$10,000 = \$5,208.75$ Yes, it's profitable with a 52% total return."

- Benefit: Complex problem becomes manageable through systematic decomposition

Basic Chain-of-Thought (CoT) Prompting

Transparency in Reasoning

Basic Prompt:

- Should we launch the product in Q2?"
- Issue: Binary answer without rationale.

CoT-Enhanced Prompt:

- Should we launch the product in Q2? Please explain your reasoning step-by-step, considering market conditions, competition, and readiness."
- Advantage: Makes the decision-making process explicit and auditable, revealing assumptions and logic gaps.

Zero-Shot Chain-of-Thought (CoT) Prompting

"Let's think step by step" Approach

Standard Zero-Shot:

- Prompt: "A farmer has 17 sheep. All but 9 die. How many are left?"
- Response: "8 sheep"
- Problem: Misinterprets "all but 9" leading to incorrect subtraction ($17-9=8$).

Zero-Shot CoT:

- Prompt: "A farmer has 17 sheep. All but 9 die. How many are left? Let's think step by step."
- Response: "Let's think step by step: The farmer starts with 17 sheep 'All but 9 die' means 9 sheep survive Therefore, 9 sheep are left"
- Benefit: The magic phrase triggers systematic reasoning, catching the linguistic nuance..

Show thinking ^

Decoding the Riddle's Logic

I'm focusing on the riddle's core elements. The question is deceptively simple. I'm carefully deconstructing the user's uploaded riddle: "A farmer has 17 sheep. All but 9 die. How many are left?" I'm meticulously analyzing each phrase. I now understand the importance of focusing on the word "but" and its role in the riddle's structure. It's a key to finding the answer.

Pinpointing the Deception

I've been meticulously dissecting the riddle, zeroing in on the deceptive wording. The phrase "All but 9 die" has revealed itself as the pivotal element. I now see the intended trap: the user's focus is cleverly diverted from the remaining sheep. I'm forming a precise, clear explanation that highlights the trickery.

There are 9 sheep left. 🐑

The key to this riddle is the phrase "All but 9 die," which means that all of the sheep died except for nine of them. Therefore, 9 sheep are still alive.

Zero-Shot Chain-of-Thought (CoT) Prompting

Applications and Limitations

Strong Applications :

- Mathematical word problems: Arithmetic reasoning improves by 40-60% with Zero-Shot CoT
- Logic puzzles and riddles: Reduces hasty pattern matching
- Multi-step instructions: Ensures sequential processing
- Common sense reasoning: Helps catch counterintuitive answers

Key Limitations

- Domain expertise required: "Let's think step by step about quantum chromodynamics" won't magically create expertise
- Inconsistent with very simple tasks: Can overcomplicate "What's $2+2$?"
- Language-dependent: Works best with "Let's think step by step" in English; translations vary in effectiveness
- Not suitable for creative tasks: "Write a poem, let's think step by step" constrains

creativity

Few-Shot Chain-of-Thought (CoT) Prompting

Providing Reasoning Demonstrations

Few-Shot Without Reasoning:

- Example 1: "Roger has 5 tennis balls. He buys 2 more cans of 3 balls each. Total: 11 balls."
- Example 2: "A cafeteria had 23 apples. They used 20 for lunch. Remaining: 3 apples."
- Query: "Jack has 15 marbles. He loses 7, then finds 4. How many does he have?"
- Response: "12 marbles"
- Problem: Examples show only answers, not reasoning process. ‘

Few-Shot With Reasoning

- Example 1: "Roger has 5 tennis balls. He buys 2 more cans of 3 balls each. Let me calculate: $5 + (2 \times 3) = 5 + 6 = 11$ balls."
- Example 2: "A cafeteria had 23 apples. They used 20 for lunch. Let me calculate: $23 - 20 = 3$ apples remaining."
- Query: "Jack has 15 marbles. He loses 7, then finds 4. How many does he have?"

ReAct (Reasoning + Acting) Framework

ReAct (Reasoning + Acting) Framework

Core Concepts: Thought-Action-Observation Cycles

Traditional Approach:

- Query: "What's the current price of AAPL stock multiplied by 100 shares?"
- Response: "Based on my last training data, AAPL was around \$150, so approximately \$15,000"
- Limitation: Outdated information, no verification possible

ReAct Approach:

- Query: "What's the current price of AAPL stock multiplied by 100 shares?"
- Thought: I need to get the current AAPL price first
- Action: SEARCH[AAPL current stock price]
- Observation: AAPL is trading at \$182.45
- Thought: Now I can calculate 100 shares value
- Action: CALCULATE[182.45 * 100]
- Observation: Result is \$18,245
- Benefit: Real-time data with transparent reasoning and verification

ReAct (Reasoning + Acting) Framework

Integration with External Tools

Static Reasoning:

- "The population of Tokyo is approximately 14 million" (potentially outdated)

ReAct with Tools:

- Thought: Need current Tokyo population
- Action: `API_CALL[world_bank_api, "Tokyo population 2024"]`
- Observation: 13,960,236 (as of 2024)
- Advantage: Accurate, verifiable, current information through tool integration

ReAct (Reasoning + Acting) Framework

Dynamic Problem-Solving

Linear Approach:

- Attempts to solve entire problem in one pass, fails if any step is wrong

ReAct Adaptivity:

- Thought: Initial approach failed, let me try alternative method
- Action: RETRY[different_strategy]
- Observation: New result obtained
- Strength: Self-corrects and adapts based on intermediate observations

Prompt Chaining

Sequential Chaining

Linear Prompt Sequences

Single Prompt Approach:

- Prompt: "Analyze this customer review, extract sentiment, identify key issues, and draft a response."
- Response: Generic analysis trying to do everything at once
- Problem: Each task gets superficial treatment, quality degrades with complexity.

Sequential Chain:

- Prompt 1: "Extract the main complaint from this review: [review text]"
 - Output: "Delayed shipping and damaged packaging"
- Prompt 2: "Analyze the sentiment of these complaints: [previous output]"
 - Output: "Highly frustrated, considering switching vendors"
- Prompt 3: "Draft an empathetic response addressing: [previous outputs]"
- Benefit: Each step gets focused attention, building on refined previous outputs.

Conditional Chaining

Decision-Based Routing

Static Chain:

- All inputs follow: Analyze → Categorize → Respond
- Problem: Simple queries waste resources, complex ones get insufficient processing.

Dynamic Routing:

- Initial Assessment: "Classify complexity: [input]" → If SIMPLE: Route to single-step response → If MODERATE: Route to 2-stage analysis → If COMPLEX: Route to multi-stage deep analysis → If UNCLEAR: Route to clarification branch.
- If-Then-Else Structures: Prompt 1: "Is this request about technical support? (YES/NO)"
- Dynamic Path Selection: Sentiment Analysis → Output: "NEGATIVE with legal threats" Dynamic Selection: - Standard path: Draft standard response - Selected path: Route to legal review → Get approved language → Draft response
- Error Handling and Fallbacks

Parallel Chaining

Concurrent Prompt Execution

Sequential Processing:

- Time 0-2s: Analyze financial data, Time 2-4s: Analyze market trends, Time 4-6s: Analyze competitor data. **Total: 6 seconds.**

Parallel Processing:

- Time 0-2s: [Simultaneously]
 - — Thread 1: Analyze financial data
 - | — Thread 2: Analyze market trends
 - | — Thread 3: Analyze competitor data
- Time 2-3s: Aggregate results Total: 3 second

Advanced Patterns

Loop Structures

Fixed Iteration:

- for i in range(3):
 - Prompt: "Improve this summary: [previous_version]"
 - Output: [refined_version]
- Problem: May over-refine or under-refine.

Conditional Loop:

```
quality_score = 0
max_iterations = 5

while quality_score < 0.8 and iterations < max_iterations:
    Prompt: "Improve this draft: [current_version]"
    Prompt: "Rate quality 0-1: [new_version]"

    if quality_score >= 0.8:
        break
    if no_improvement_detected:
        break
```

Advanced Patterns

Recursive Prompting

Simple Recursion:

```
Function: Explain(concept, depth):  
  if depth == 0:  
    return basic_explanation  
  else:  
    explanation = Explain(concept, depth-1)  
    Prompt: "Elaborate on: [explanation]"  
    return enhanced_explanation
```

Example: Explain("recursion", 3)
→ Basic → Detailed → Expert-level explanation

Tree Recursion:

Break Complex Problem:

"Plan a product launch"



Advanced Patterns

Self-Correction Chains

Single-Pass Verification:

- Generate → Verify once → Output
- Limitation: May miss subtle errors.

Multi-Layer Verification:

Code Generation Pipeline:

1. Generate: Create Python function
2. Syntax Check: "Parse this code for syntax errors"
3. Logic Check: "Identify logical flaws"
4. Test Generation: "Create test cases"
5. Test Execution: Run tests (simulated)
6. Performance Review: "Analyze efficiency"

If any check fails:

- Loop back to generation with error context
- Maximum 3 regeneration attempts
- Fallback to simpler solution if needed

Advanced Patterns

Verification Loops

Without Self-Correction:

- Generate output → Deliver to user
- Risk: Errors propagate unchecked.

With Self-Correction:

Step 1: Generate initial response

Step 2: "Identify potential errors in: [response]"

Step 3: "Correct these issues: [errors] in [response]"

Step 4: "Verify corrections: [corrected_response]"

Example Flow:

Initial: "The capital of Australia is Sydney"

Error Detection: "Sydney is not the capital"

Correction: "The capital of Australia is Canberra"

Verification: "Confirmed: Canberra is correct"

LLM Settings in Prompt Engineering

LLM Settings in Prompt Engineering

Introduction to control

When designing and testing prompts, control settings on the LLM API are crucial for fine-tuning the AI's response. These settings adjust the AI's behavior to produce more reliable and desirable responses based on specific use cases.

- Example
 - Temperature
 - Top P (Nucleus Sampling)
 - Max Length
 - Stop Sequences
 - Frequency Penalty
 - Presence Penalty

LLM Settings in Prompt Engineering

Temperature control

Controls the randomness of the AI's responses.

- Low Temperature results in more deterministic and factual responses.
 - It is ideal for tasks like fact-based Q&A.
- High Temperature increases creativity and diversity in responses.
 - It is suitable for tasks like poetry or creative writing.

LLM Settings in Prompt Engineering

Temperature control

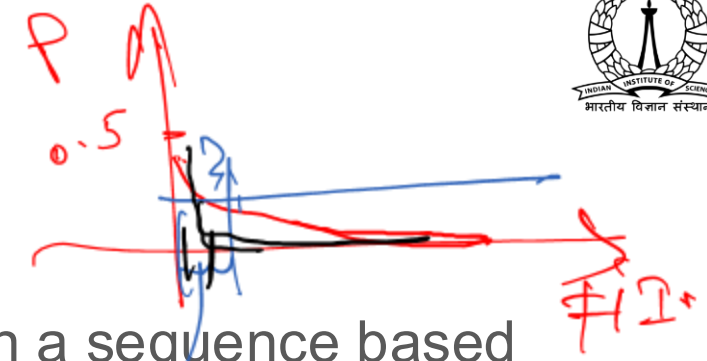
- Correct Example:
 - Setting: Temperature at 0.3
 - Prompt: "Write a brief summary of the latest UN climate change report."
 - *Explanation: A low temperature setting is appropriate here to ensure factual accuracy and a coherent summary based on a reliable interpretation of data.*
- Incorrect Example:
 - Setting: Temperature at 0.9
 - Prompt: "Write a brief summary of the latest UN climate change report."
 - *Explanation: A high temperature for this task may produce a summary with irrelevant or creatively interpreted facts, which is undesirable for accurate reporting.*

LLM Settings in Prompt Engineering

Temperature control

- Correct Example:
 - Prompt: "Write a poem about the ocean."
 - Temperature Setting: High (to encourage more creative and abstract outputs).
 - *Explanation: A higher temperature setting here promotes creativity, which is suitable for poetic expressions.*
- Incorrect Example:
 - Prompt: "Write a step-by-step guide on how to file taxes."
 - Temperature Setting: High.
 - *Explanation: A high temperature for this task might produce creative but impractical and incorrect tax advice. Lower temperature is preferred for factual accuracy and coherence.*

LLM Settings in Prompt Engineering

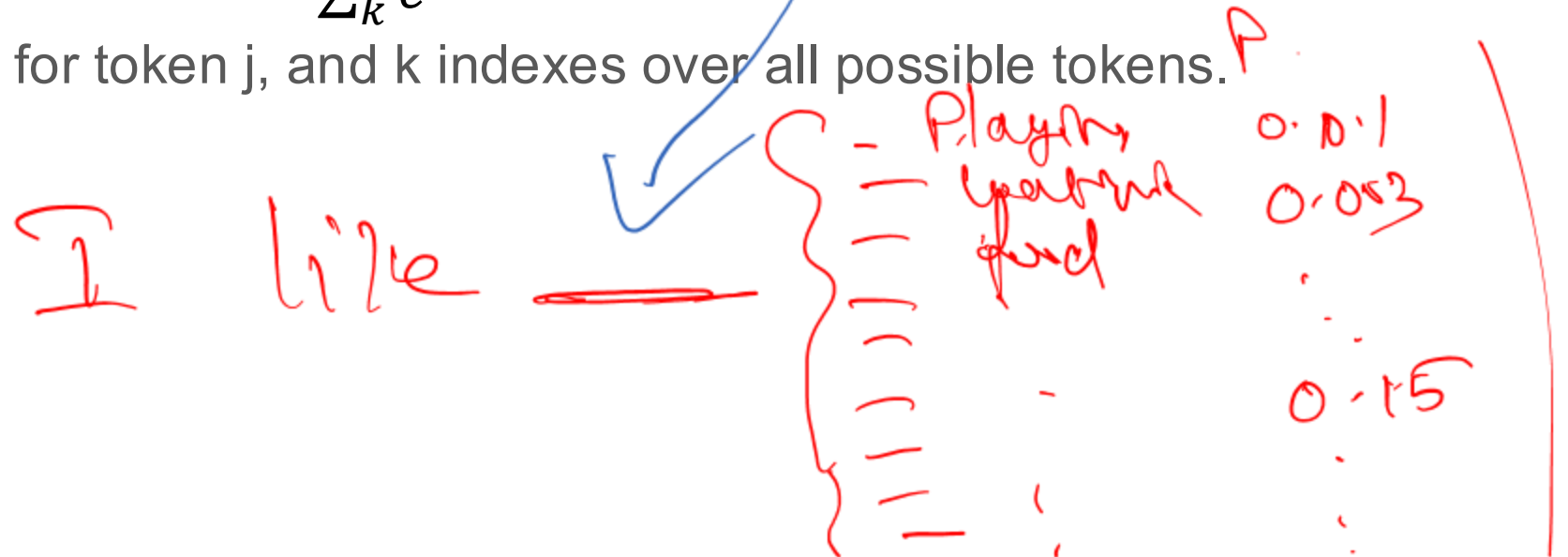


Temperature control

- LLMs predict the next token (e.g., word or punctuation) in a sequence based on the probability distribution derived from the training data.
- The probabilities for each token are computed using a softmax function, which converts the raw output scores (logits) from the model into probabilities.
- The softmax function for a token “j” is calculated as follows

$$P(j) = \frac{e^{z_j}}{\sum_k e^{z_k}}$$

where z_j is the logit for token j, and k indexes over all possible tokens.



LLM Settings in Prompt Engineering

Temperature control

- Temperature modifies the softmax function $P(j) = \frac{e^{\frac{z_j}{T}}}{\sum_k e^{\frac{z_k}{T}}}$
- $T > 1$: This makes the softmax distribution flatter.
- $T < 1$: This sharpens the softmax distribution, making the highest logit significantly more pronounced compared to others.
- For logits [2.5, 1.0, 0.5]:
 - At $T = 1$: Softmax might yield probabilities like [0.70, 0.20, 0.10].
 - At $T = 2$: The flatter softmax might yield probabilities like [0.50, 0.30, 0.20].
 - At $T = 0.5$: The sharper softmax might yield probabilities like [0.85, 0.10, 0.05], heavily favoring the first token.

<https://colab.research.google.com/drive/1EAKB4mbiB0RFPSyhJKn8X2000Ax-UsAB?usp=sharing>

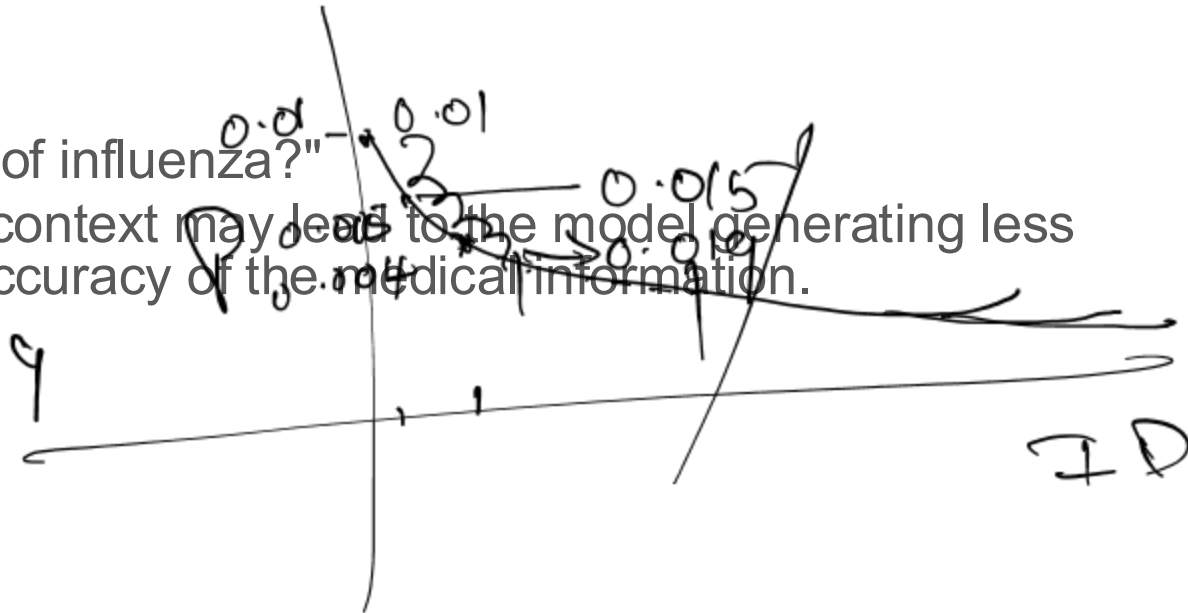
LLM Settings in Prompt Engineering

Top_P (Nucleus Sampling)

Controls the breadth of possible responses by only considering the top p probability mass.

- Correct Example:
 - Setting: Top P at 0.2
 - Prompt: "What are the main symptoms of influenza?"
 - Explanation: A low Top P setting is used to ensure that the response is focused and factual, drawing from the most likely tokens related to medical accuracy.
- Incorrect Example:
 - Setting: Top P at 0.9
 - Prompt: "What are the main symptoms of influenza?"
 - Explanation: A high Top P value in this context may lead to the model generating less likely symptoms, reducing the factual accuracy of the medical information.

Top_P = 0.019



LLM Settings in Prompt Engineering

Top_P (Nucleus Sampling)

1. **Compute Probabilities:** First, the model calculates the softmax probabilities for each token in the vocabulary based on the logits output by the model.
2. **Sort Probabilities:** The tokens are then sorted by their probability, from highest to lowest.
3. **Cumulative Probability Calculation:** The cumulative probability is computed for the sorted list of tokens. This process continues until the sum of the probabilities reaches or exceeds the threshold specified by `top_p`.
4. **Token Selection:** Only those tokens that are part of the cumulative sum that meets or exceeds `top_p` are considered for selection. The actual token is then randomly sampled from this reduced set according to their probabilities.

LLM Settings in Prompt Engineering

Top_P (Nucleus Sampling)

- Example of Top_p Calculation
 - Suppose the model produces the following probabilities for a set of tokens:
 - Token A: 0.4 Token B: 0.3 Token C: 0.2 Token D: 0.1
 - If top_p is set to 0.7, the tokens are considered in the following way:
 - Calculate cumulative probabilities as we add each token starting from the most probable:
 - Token A (0.4), Token A + Token B ($0.4 + 0.3 = 0.7$)
 - Since adding Tokens A and B reaches the cumulative probability of 0.7, Tokens C and D are excluded from the selection pool, even if their combined probabilities would not substantially exceed the threshold.

LLM Settings in Prompt Engineering

Temperature & Top_P

What to Expect from These Settings?

- High Temperature with Low Top_p
- Low Temperature with High Top_p

Generative and Agentic AI in Practice: DS 246 (1:2)

Topics Covered

Phase 1: Foundations (Weeks 1-8, leading to Midterm)

- Week 2 (Aug 13): Generative AI Core Concepts & Architecture
 - Topics: Prompting Principles, Patterns, Techniques (Zero-Shot, CoT, ReAct), Prompt Chaining, LLM settings
 - Jailbreaking (Mon, 18th Aug)
- **Assignment (Mon)**
 - Achieve a target output using different prompting techniques and compare performance