Foundational Concept: LLMs are Prediction Engines
Before diving into techniques, remember the core idea: LLMs predict the next token based on the input sequence and their training data. Prompt engineering is about crafting that input sequence to guide the LLM towards the desired prediction, often by encouraging a more deliberate and refined "thought" process.

Core Idea for Maximizing Accuracy: Iterative Refinement Loop (Simulated "Recursive Learning")
A powerful strategy, often leading to the best answers for complex tasks, is to guide the LLM through an **iterative refinement loop**. This involves prompting the LLM to:

1. Generate an initial thought, plan, or draft answer.
2. Critically evaluate or reflect on that initial output.
3. Revise and improve upon it based on the self-critique or new information.
   This loop can be explicitly prompted or implicitly encouraged by certain techniques (like CoT, ToT, Self-Consistency, ReAct) and can be repeated multiple times for enhanced quality. It simulates a "learning" or "thinking deeper" process within the generation.

I. LLM Output Configuration (Crucial Companion to Prompting)
While not a prompting technique itself, configuring the LLM's output parameters is essential for getting the best results from your prompts, especially when using iterative refinement which may require more tokens.

- **Output Length (Max Tokens)**
  - **What**: Sets the maximum number of tokens the LLM can generate.
  - **Why**: Controls cost, energy, response time. Prevents overly long or truncated responses. Techniques involving iterative refinement or intermediate steps (CoT, ReAct, explicit refinement loops) will need sufficient tokens for all stages.
  - **When to Use**: Always.
    - For concise answers, use a lower limit.
    - For detailed explanations, creative writing, or iterative refinement, allow more tokens.
  - **How to Use**: Set "Token Limit." The prompt can also guide length.
- **Sampling Controls: Temperature, Top-K, Top-P**

- **What**: Control how the LLM selects the next token.
- **Why**: Determine randomness/creativity vs. determinism/factuality.
- **When & How to Use**:
    - **Temperature**:
        - **What**: Controls randomness.
        - **Why**: Low for factual; high for creative/diverse.
        - **When**:
            - Temp = 0: Single correct answer, often used in the *final step* of a refined reasoning chain.
            - Low Temp (0.1 - 0.3): Factual, coherent, summarization, Q&A. Good for evaluation steps in a refinement loop.
            - High Temp (0.7 - 1.0+): Creative tasks, brainstorming, generating diverse initial thoughts for a refinement loop (see Self-Consistency).
        - **How**: Adjust slider/value.
    - **Top-K**:
        - **What**: Considers top K most probable tokens.
        - **Why**: Limits choice pool.
        - **When**: Low for restrictive/factual; High for more creative.
        - **How**: Set Top-K value.
    - **Top-P (Nucleus Sampling)**:
        - **What**: Selects from smallest set whose cumulative probability exceeds P.
        - **Why**: Balances diversity and quality.
        - **When**: High Top-P (0.9 - 0.99) often for good quality, diverse outputs.
        - **How**: Set Top-P value (0 to 1).
- **Putting it all Together**: Settings interact.

        - **For Iterative Refinement**: Initial thought generation might benefit from higher temperature/diversity, while critique and final answer generation might prefer lower temperature for focus and accuracy.
    - **Warning**: Extreme settings can make others irrelevant. Beware "repetition loop bug."

II. Core Prompting Techniques

These structure your prompt text. Many can be enhanced by incorporating an iterative refinement mindset.

- **General Prompting / Zero-Shot Prompting**
    - **What**: Simple task description without examples.
    - **Why**: Low effort, baseline.
    - **When**: Simple tasks, capable LLMs, first attempt.
    - **How**: Clearly state instruction. Be precise.
        - **With Simple Refinement**: Even in zero-shot, you can add a general refinement instruction: "Explain X. Before finalizing, review your explanation for clarity and accuracy."
    - **Example**: Classify movie reviews as POSITIVE, NEUTRAL or NEGATIVE. Review: "Her" is a disturbing study... Sentiment:
- **One-Shot & Few-Shot Prompting**
    - **What**: Provide 1 (one-shot) or 3-5 (few-shot) input/output examples.
    - **Why**: Shows desired structure, style, detail. Improves accuracy.
    - **When**: Zero-shot fails, specific format needed, pattern learning.
    - **How**:
        - Clearly delineate examples.
        - **Quality Examples**: Accurate, relevant, diverse, edge cases.
        - **Demonstrating Refinement**: If examples are complex, they could even show a mini-refinement step within the example output to teach the pattern.
        - **Classification**: Mix class examples.

- **System, Contextual, and Role Prompting**
    - A. **System Prompting**:
        - **What**: Sets overall context, purpose, high-level instructions.
        - **Why**: Guides fundamental behavior, enforces formats, safety.
        - **When**: Consistent output structure, enforce constraints, define overall "mode."
        - **How**: Beginning of prompt or separate field. Be explicit.
            - **System Prompt for Refinement**: "You are a meticulous assistant. For any complex query, first outline your thought process, then generate an answer, then critically review your answer for errors or omissions, and finally provide the polished result."
        - **Example (JSON output)**: ...return valid JSON.
    - B. **Role Prompting**:
        - **What**: Assigns a persona or expertise.
        - **Why**: Tailors tone, style, knowledge.
        - **When**: Emulate professions, specific styles, creative writing.
        - **How**: "I want you to act as a [Role who inherently self-corrects, e.g., 'Senior Editor', 'Fact-Checker']. ..."
    - C. **Contextual Prompting**:
        - **What**: Provides specific background info.
        - **Why**: Nuance, tailored responses, reduces ambiguity.
        - **When**: Ongoing conversations, tasks requiring specific non-obvious knowledge.
        - **How**: Prefix task with relevant context. Context often *results* from a previous step in an iterative refinement loop.
- **Step-Back Prompting**
    - **What**: Two-step: 1) Ask a general/principle-based question. 2) Use its answer as context for the specific task.

    - **Why**: Activates background knowledge, encourages critical thinking, applies principles. This is an explicit two-step refinement process.
    - **When**: Complex tasks, direct prompting yields superficial results, mitigate bias.
    - **How**:
        - **Step 1 (Initial Abstraction/Thought)**: Ask a general conceptual question.
        - **Step 2 (Refinement & Application)**: Use Step 1's output as context for the specific prompt.
- **Chain of Thought (CoT) Prompting**
    - **What**: Prompts LLM for intermediate reasoning steps before the final answer.
    - **Why**: Improves reasoning, provides interpretability, robust. This is the foundation of many explicit refinement techniques.
    - **When**: Multi-step reasoning, direct answers fail, need to understand LLM's "thinking."
    - **How**:
        - **Zero-Shot CoT**: Append "Let's think step by step."
            - **Enhance with Self-Correction**: "Let's think step by step. Critically review each step for accuracy before proceeding to the next."
        - **Few-Shot CoT**: Provide examples including reasoning steps and final answer. Examples can demonstrate self-correction within the chain.
        - **CoT Best Practices**: Final answer after reasoning. Temp=0 for greedy, focused reasoning path, *unless* deliberately seeking diverse paths (see Self-Consistency).
- **Self-Consistency**
    - **What**: Extends CoT. Prompt multiple times with same CoT prompt (high temp for diverse paths). Extract final answers. Choose most common (majority-voted).

    - **Why**: Improves accuracy by exploring multiple reasoning paths and selecting the most robust outcome. An explicit form of iterative refinement through parallel exploration and consensus.
    - **When**: Complex reasoning, high accuracy needed, cost acceptable, ambiguous tasks.
    - **How**: Use CoT prompt. Run multiple times (e.g., 3-5+). High temp. Extract answers. Majority vote. This is "refinement through diversity and voting."
- **Tree of Thoughts (ToT)**
    - **What**: Generalizes CoT. LLM explores multiple reasoning paths (tree branches), generates intermediate "thoughts" at each step, evaluates them, and decides which paths to explore/prune.
    - **Why**: Better for complex tasks needing exploration, lookahead, planning. Allows systematic exploration and self-correction. A sophisticated form of explicit, structured iterative refinement.
    - **When**: Very complex problems (game playing, proofs), tasks where deliberate exploration and evaluation are key.
    - **How**: Advanced, often needs a framework. Prompt might be: "Generate 3 possible next thoughts...", then "Evaluate these thoughts...", then "Select the best thought and elaborate...". This explicitly models the "generate-evaluate-refine" loop.
- **ReAct (Reason & Act)**
    - **What**: LLM combines natural language reasoning with external tools (search, APIs) via a thought-action-observation loop.
    - **Why**: Overcomes LLM limits (real-time info, calculations, external interaction).
    - **When**: Up-to-date info, fact-checking, calculations, API interaction.
    - **How**: Needs framework (e.g., LangChain).
        - **Loop (inherently iterative refinement)**:

1. **Thought**: LLM reasons, plans action.
2. **Action**: LLM/framework executes tool action.

3. **Observation**: Tool result fed back.
    - LLM uses observation to **refine its understanding and generate the next thought/action**, repeating until a final answer is derived. Each cycle is a refinement.

- **Automatic Prompt Engineering (APE)**
    - **What**: Using an LLM to generate/refine prompts.
    - **Why**: Reduces manual effort, potentially finds better prompts.
    - **When**: Need many prompt variations, explore prompt structures, have evaluation metrics.
    - **How**:
        - **Generation Prompt**: Instruct LLM to generate prompt candidates.
        - **Evaluation**: Score generated prompts.
        - **Selection/Refinement**: Choose best. This whole process is an iterative refinement loop applied to the prompts themselves.
- **Code Prompting**
    - All code-related prompting can benefit immensely from an iterative refinement loop:
        - A. **Prompts for Writing Code**: "Write the code. Then, review the code for potential bugs or inefficiencies and provide a revised version with explanations."
        - B. **Prompts for Explaining Code**: "Explain this code. Are there alternative interpretations? Provide the most likely explanation with your reasoning."
        - C. **Prompts for Translating Code**: "Translate this code. Then verify if common idioms from the source language were translated effectively to the target language and refine if necessary."
        - D. **Prompts for Debugging and Reviewing Code**: "Debug this code. Explain the bug. Suggest a fix. Then, suggest further improvements for robustness and readability."

III. General Best Practices (Applicable across most techniques)

1. **Embrace Iterative Refinement**: For any non-trivial task, explicitly prompt the LLM to generate an initial output, then review and refine it.
    - **How**: "First, provide a draft of X. Then, act as a critic and identify flaws or areas for improvement in your draft. Finally, provide a revised, final version incorporating those improvements."
    - This strategy leverages the LLM's ability to "think" more deeply by breaking down the problem.
2. **Provide Examples (Reiteration of One/Few-Shot)**: Highly effective. Examples can demonstrate a refinement process.
3. **Design with Simplicity (for each step of refinement)**: Clear, concise instructions *for each stage* of the iterative loop are easier to understand.
4. **Be Specific About the Output (at each stage and for the final product)**: Detail desired format, length, content, style.
5. **Use Instructions over Constraints**: Tell the model what to do at each refinement stage.
6. **Control the Max Token Length**: Iterative processes consume more tokens.
7. **Use Variables in Prompts**: Makes multi-step/iterative prompts reusable.
8. **Experiment with Input Formats and Writing Styles**: Try different ways to phrase instructions for generation, critique, and revision.
9. **For Few-Shot Prompting with Classification Tasks, Mix Up the Classes**.
10. **Adapt to Model Updates**: New models might handle iterative instructions better or differently.
11. **Experiment with Output Formats (e.g., JSON)**: Can be particularly useful for structured intermediate thoughts in a refinement loop.
    - **JSON Repair**: Important if an intermediate step in a JSON-based refinement process truncates.
12. **Working with Schemas**: Define schemas for inputs and expected intermediate/final outputs to guide structured refinement.
13. **Experiment Together with Other Prompt Engineers**: Share iterative refinement strategies.

14. **CoT Best Practices (for each reasoning phase within an iteration)**: Answer after reasoning; set temperature to 0 for focused refinement, or higher if exploring alternatives for a critique phase.
15. **Document the Various Prompt Attempts (Especially Iterative Structures)**:
    - **Crucial for iterative refinement**: Track the full sequence of prompts and responses in your loops.
    - Use a structured format. Track: Prompt Name/Version, Goal, Model, Parameters, Full Prompt Text for *each step* (initial, critique, revise), Full Output for *each step*, Result, Notes.