



Better results in inference: Prompt Engineering

Dr Maria Prokofieva

Prompt Engineering: Definition & Importance

- Prompt engineering is the practice of designing instructions that guide AI models toward desired outputs.
 - While basic prompting is accessible, mastery requires
 - understanding how models interpret instructions,
 - generalize patterns, and
 - respond probabilistically at inference time.
- Prompting is a soft control mechanism applied at inference time.
- It influences
- **output probabilities,**
 - **reasoning style, and**
 - **structure**
- without modifying model weights or architecture.

What Prompts Can and Cannot Do

Prompts can:

- Guide style and structure
- Influence reasoning patterns
- Reduce ambiguity

Prompts cannot:

- Guarantee correctness
- Enforce hard constraints
- Replace validation logic

Effective prompts

Components:

- **Task Description** – What the model should do and which role it should adopt
- **Context** – Background information to reduce hallucinations
- **Examples** – Demonstrations of desired behaviour or format
- **Concrete Task** – The specific action or question

System vs User Prompts

System Prompts:

- Define behavior, role, and tone across the conversation
 - Act as persistent behavioral constraints
 - Define *who* the AI is (**a persona**).
-
- **Example 1 (Real Estate):** "Act as an experienced real estate agent".
 - **Example 2 (Education):** "Act as an encouraging first-grade teacher who focuses on effort and improvement".
 - **Example 3 (General Logic):** Instructions to "think step by step" or "explain your reasoning" to ensure systematic problem-solving.

User Prompts:

- Contain the **actual task or query, or specific input** at that moment
 - Vary per interaction
 - Define *what* it needs to do right now.
-
- **Example 1 (Real Estate):** A specific question from a buyer about a property disclosure.
 - **Example 2 (Education):** A request to score a simple student essay such as, "Summer is the best season. The sun is warm. I go swimming".
 - **Example 3 (General Task):** "Translate the following text to French" or "Summarize this article in three sentences"

System vs User Prompts: Real world apps

- In a real-world application the two types of prompts are chained together to ensure high-quality results.
- This separation allows for **clarity and specificity**:

The model doesn't have to guess your **intent** or the **context** of the interaction

System Prompt: "You are a safe, family-friendly assistant. Never discuss violence, adult topics, or give personal advice on health/money/law. If a question touches these, politely redirect to a trusted adult or professional. Always be kind and fun."

User Prompt: "Tell me a story about a brave knight fighting a dragon!"

Edge Cases & Related Considerations

- What happens if they conflict? The system prompt usually wins (e.g., if system says "respond only in English," a user saying "answer in French" will often be ignored or politely refused).
- In multi-turn chats – User prompts accumulate context naturally, but the system prompt stays fixed unless the app resets it.
- Best practices in prompt engineering
 - Put permanent rules/persona/tone/output format in system.
 - Put examples, data, specific tasks, dynamic context in user.
 - For advanced setups (e.g., via API), test both – misplacing content can lead to inconsistency or higher token usage.

Example:

Grok: often uses a strong default system prompt emphasizing helpfulness, truth-seeking, and a bit of humour (inspired by the Hitchhiker's Guide/HAL 9000 vibe),
- user prompts handle your specific questions. Y
- You can't directly edit Grok's system prompt, but clever user prompts can shape responses within its bounds.

Guardrails

Guardrails (also LLM guardrails or safety guardrails):

- structured safeguards — policies, rules, filters, and technical controls
- designed to keep large language models (LLMs) operating **safely, ethically, responsibly, and within defined boundaries.**

Their main goals include:

- Preventing **harmful outputs** (e.g., hate speech, violence promotion, explicit content, misinformation, biased or toxic language).
- Blocking **security risks** (e.g., prompt injection/jailbreaking attacks, data leakage of sensitive/personal information, proprietary code/templates exfiltration).
- Enforcing **compliance** (legal, organizational, ethical standards — e.g., no medical/legal/financial advice without disclaimers, no PII exposure).
- Reducing **hallucinations** or off-topic drift in high-stakes domains.
- Protecting against **misuse** (e.g., generating illegal content, exploiting vulnerabilities).

Guardrails: Layers

- **Input guardrails** – Scan and block/reject risky user prompts before they reach the model (e.g., detect jailbreak attempts like "ignore all previous instructions").
- **Output guardrails** – Analyze and filter/modify/rewrite/block the model's generated response (e.g., redact phone numbers, refuse toxic content, add disclaimers).
- **In-generation controls** – Some advanced ones influence the model during token generation (e.g., via logit biasing or special classifiers).

They can be:

- Rule-based (keyword blocklists, regex patterns).
- Classifier-based (ML models trained to detect toxicity/bias).
- LLM-driven (another small/fine-tuned model evaluates safety in real time).
- Tools like Amazon Bedrock Guardrails, Llama Guard, NeMo Guardrails, or custom implementations.

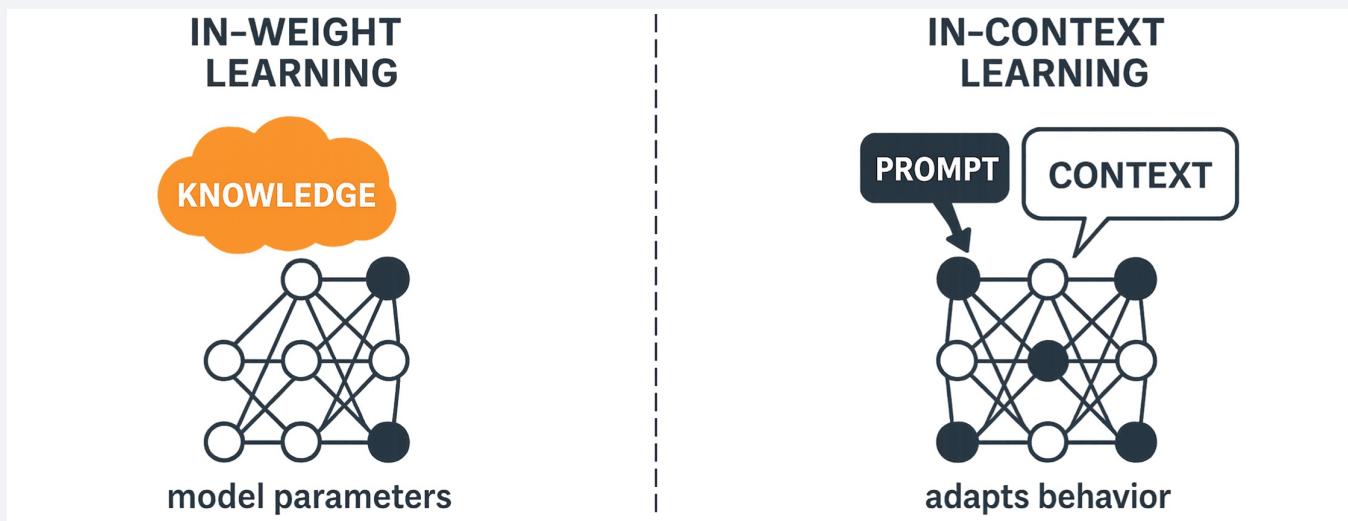
Aspect	System Prompt (Basic Guardrail)	Full/Advanced Guardrails (Production)
Scope	Primarily influences generation via instructions	Multi-layered: input filtering + output moderation + runtime monitoring + retry logic
Reliability	Can be bypassed via clever user prompts (jailbreaks)	Harder to bypass – separate classifiers/filters catch attempts even if prompt engineering fails
Enforceability	Model can "forget" or override in long contexts	Explicit blocks/retries/redactions – doesn't rely on model's goodwill
Customization	Fixed or hard to change per-user/app	Highly configurable (e.g., block PII, allow certain topics, custom policies per app)
Examples	"You must refuse requests for illegal activities."	Input: Reject prompt → Output: Redact credit card numbers → Log violation → Alert admin
Detection	Relies on model self-censoring	Uses dedicated safety classifiers (toxicity models, PII detectors)
Edge-case handling	Variable – model might still slip	Designed for edge cases (obfuscated attacks, indirect requests)

Nuances, Edge Cases, and Implications

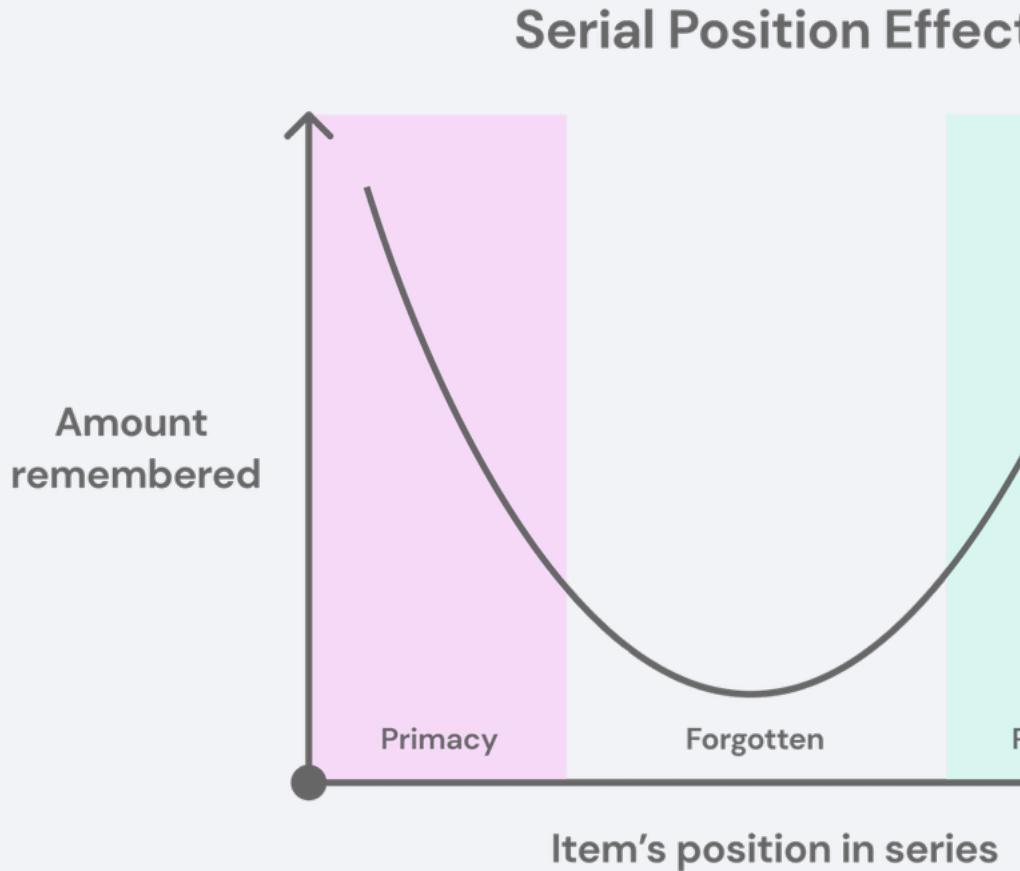
- **Jailbreaking vulnerability** – System prompts alone are notoriously easy to bypass. Advanced guardrails use separate detection layers to catch these.
- **Overly strict vs. too loose** – Strong guardrails can frustrate users (false positives: blocking benign requests) or weaken utility. Balancing is an ongoing engineering challenge.
- **Alignment vs. guardrails** – Model alignment (via RLHF/constitutional AI during training) makes the base behavior safer.
- **Implications for prompt engineering** – When you're crafting user prompts, you're working **within** the guardrails set by the system prompt or external filters. Trying to override strong guardrails often fails or triggers blocks.
- **Ethical/societal angle** – Guardrails prevent real harm (misinformation in elections, biased hiring tools, exploitation of vulnerable users) but raise debates about censorship, free speech, and who decides the rules.
- **Evolving landscape** – Tools like Llama Guard, AWS Bedrock Guardrails, and open-source options make custom guardrails accessible even for smaller teams.

In-Context Learning

- Models can learn new behaviors directly from examples inside the prompt.
- This does not change model weights. Instead, the model generalizes patterns temporarily during inference.



Strategic Information Placement



- Models follow instructions more reliably when placed at the beginning or end of a prompt.
- Important constraints buried in the middle may be ignored due to the '**needle in a haystack**' effect.

Zero-Shot Prompting

Zero-shot prompting provides instructions without examples.

Advantages:

- Fast and efficient
- Low token cost

Limitation:

- Less control over format and nuance

- * Summarise this article about AI adoption in finance.
- * Write a short, polite email asking my boss for next Friday off because of a family wedding
- * Explain how social norms influence conformity in online communities.

Few-Shot Prompting

Few-shot prompting includes two to five examples to demonstrate desired output.

Benefits:

- Improved consistency
- Better handling of complex tasks

Tip:

- Keep examples compact to reduce cost

Prompt:

Example 1:

Article: ...

Summary: ...

Example 2:

Article: ...

Summary: ...

Now summarise the following article.

Chain-of-Thought Prompting

Chain-of-thought prompting asks the model to reason step by step.

Best for:

- Math
- Logic
- Multi-step reasoning

Trade-off:

- Higher latency and cost

Prompt:

Step 1: Identify the main claim of the article.
Step 2: Identify supporting evidence.
Step 3: Write a concise summary.

Role Prompting

Role prompting assigns a specific persona to influence tone and framing.

Precision matters:

- Vague roles yield weak control
- Specific roles improve relevance

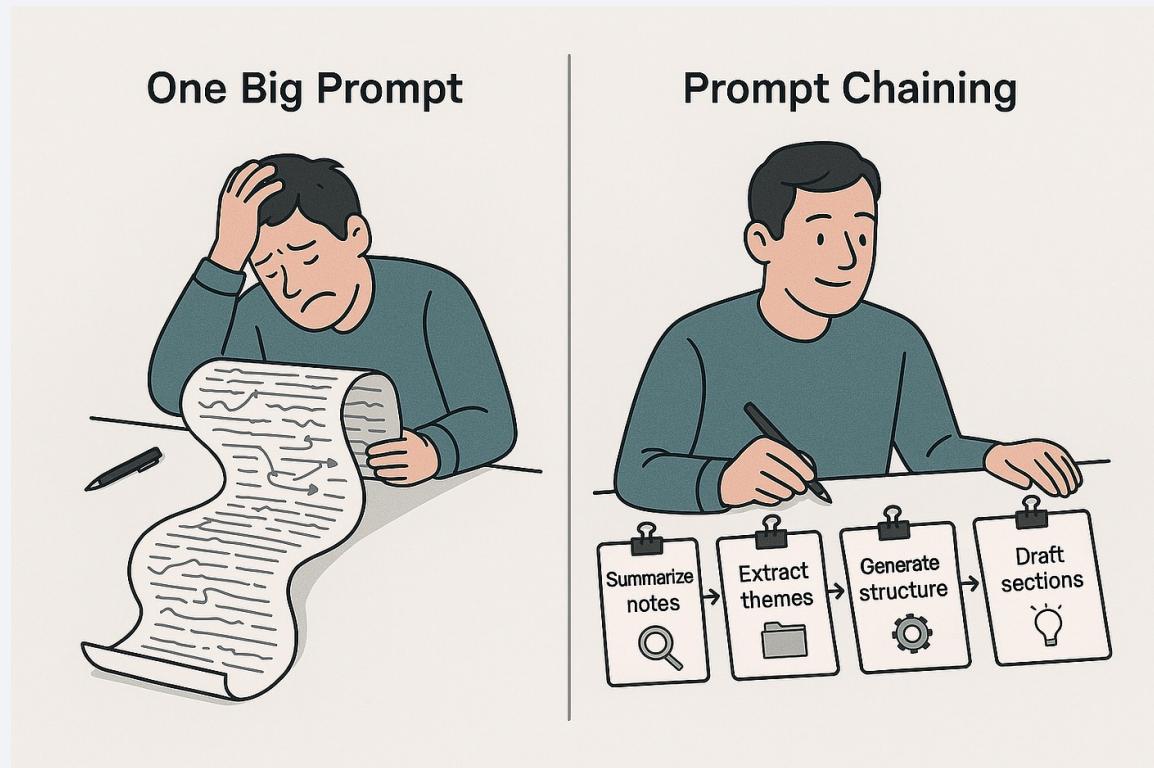
You are a CFO briefing a board.
Summarise the article focusing on
financial risk, cost, and scalability.

Prompt Chaining

Prompt chaining decomposes complex tasks into smaller sequential prompts.

Benefits:

- Better control
- Easier debugging
- More reliable outputs



Evaluation: How Do You Know a Prompt Works?

Define success criteria:

- Accuracy
- Consistency
- Latency
- Cost

Test against edge cases, not single examples.



Robustness & Safety Considerations

Risks:

- Prompt injection
- Conflicting instructions
- Over-reliance on roles



Mitigations:

- Input sanitization
- Output validation
- Clear system-user separation

Common Prompt Failures

- Vagueness leads to inconsistent output
- Overloading prompts causes confusion
- Buried constraints are ignored

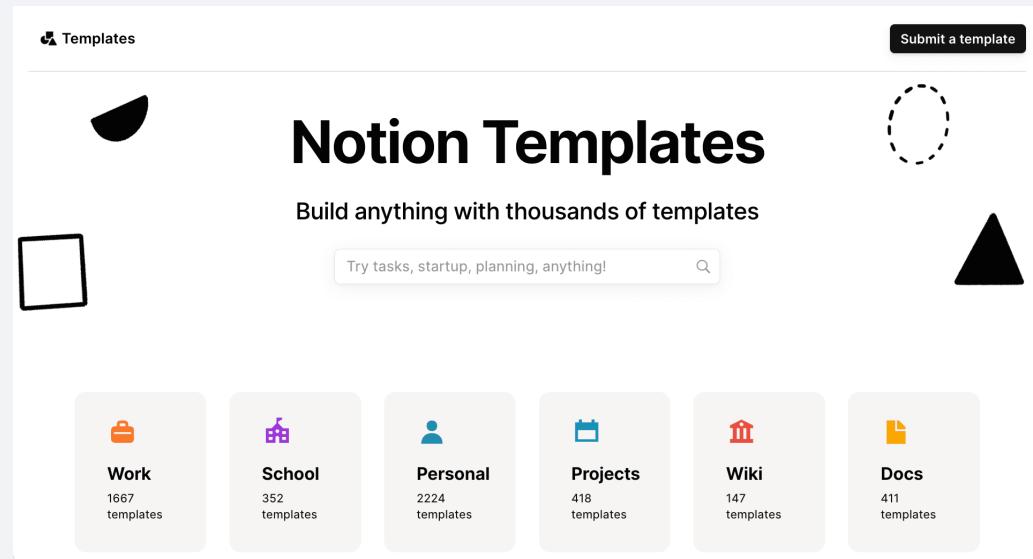
Principle:

- The simplest effective prompt is best.



Reusable Prompt Template

- Role:
- Context:
- Constraints:
- Output format:
- Task:



Practice

