Guide to Prompt Engineering Frameworks

Date: December 11, 2025

1. Introduction

Prompt engineering has evolved from an experimental art into a structured discipline essential for maximizing the performance of Large Language Models (LLMs). For data scientists, utilizing defined frameworks ensures reproducibility, scalability, and precision in model outputs. This document provides a detailed reference for the most effective prompt engineering frameworks available.

---

2. Frameworks Reference

1. COSTAR (Context, Objective, Style, Tone, Audience, Response)

Best For: Structuring high-stakes, professional outputs where nuance and format are critical. **Scenario:** A Lead Data Scientist needs to generate a **Model Governance Report** for a new credit scoring model that uses "black box" features. This report must convince the Risk Management Committee (non-technical) that the model is safe, compliant, and superior to the old logistic regression.

Framework Components

The COSTAR framework consists of six interconnected elements:

Context (C): Provide comprehensive background information that grounds the AI in your specific situation. This eliminates irrelevant outputs by establishing what domain, constraints, and prior knowledge apply. For instance, mentioning that you're working with healthcare data versus social media data fundamentally changes how the model should approach the task. Context acts as the "reality check" that prevents the model from operating in a vacuum.

Objective (O): Define the explicit goal and what success looks like. Rather than asking "summarize this," specify "identify the three most critical risks and rank by financial impact." This clarity prevents task drift and ensures the model stays focused on your actual need, not a tangential interpretation. Clear objectives are particularly valuable for models that might otherwise generate comprehensive but ultimately unhelpful responses.

Style (S): Specify the structural approach and organizational format. Should the response be technical and detailed with subsections? Conversational and narrative? Bulleted for quick scanning? The style determines how information is organized and presented, making the output immediately usable in your workflow without reformatting.

Tone (T): Set the emotional quality and voice of the response. This can range from formal and authoritative (useful for compliance documents) to friendly and encouraging (useful for user-facing materials). Tone shapes how the audience perceives the information, influencing trust, engagement, and receptivity. A risk report in an encouraging tone would seem dismissive of actual risks, while a friendly onboarding guide in a formal tone would feel cold and unwelcoming.

Audience (A): Identify who will consume this output and what their knowledge level is. Tailor vocabulary, depth of explanation, and complexity accordingly. An explanation for C-level executives differs dramatically from one for engineers. The same technical fact needs different framing, examples, and context depending on whether your audience understands gradient descent or needs to understand business impact.

Response (R): Define the exact format and structure of the output—JSON, CSV, Markdown sections, a specific XML schema, or something else. This is critical for integration into systems or handoff workflows. Specifying "provide a JSON object with keys 'risk', 'mitigation', 'timeline'" prevents the AI from giving you prose that must be manually restructured.

Why COSTAR is Powerful

COSTAR forces systematic thinking before you even write the prompt. It reduces hallucinations by giving the model firm guardrails. It scales well: teams can standardize on COSTAR prompts across projects, making outputs predictable and comparable. It's especially effective for high-stakes tasks where the output quality directly impacts decisions.

Complex Example Prompt

```
 # CONTEXT
We have developed a new credit risk scoring model using XGBoost, trained on 5 years of loan application data (n=2,000,000). The new

# OBJECTIVE
Write an Executive Risk Assessment Report justifying the deployment of the XGBoost model. You must acknowledge the complexity of the

# STYLE
Corporate Risk Reporting. Structured, evidence-based, and authoritative. Use clear headings and section breaks.

# TONE
Professional, transparent, yet confident. Avoid overly academic jargon; focus on business impact and risk mitigation.

# AUDIENCE
The Risk Management Committee. This includes the Chief Risk Officer (CRO), Legal Counsel, and the VP of Lending. They understand bus

# RESPONSE
Provide the response as a formal Markdown report with the following sections:
1. Executive Summary
2. Performance Benefit Analysis (Translate AUC to financial impact)
3. Interpretability & Fairness (Explain SHAP in layman's terms)
4. Fallback Mechanisms (What happens if the model is unsure)
5. Final Recommendation
```

2. RACE (Role, Action, Context, Execute)

Best For: Quick iteration and daily tasks where you need role-specific expertise applied immediately. **Scenario:** A DevOps engineer needs the AI to act as a senior Kubernetes architect to review a problematic cluster configuration and suggest fixes.

Framework Components

The RACE framework uses four core elements to produce expert-aligned outputs quickly:

Role (R): Define who the AI should be. Specify depth and specialty: "Senior DevOps Engineer with 10+ years in Kubernetes" differs from "Linux system administrator." A "Product Manager focused on B2B SaaS" approaches problems differently than a "Product Manager in fintech." The role establishes both expertise level and perspective, creating a mental model that guides all subsequent responses.

Action (A): State the specific task or operation the AI should perform. This is the verb: "analyze," "refactor," "debug," "propose," "compare." Being explicit about the action prevents the AI from defaulting to explanation or summary when you need troubleshooting or creative ideation.

Context (C): Provide relevant constraints, data, or situational details. What are the requirements? Limitations? Background facts? Context grounds the action in reality. For instance, "Debug this code with the constraint that we can only use Python 3.9 and cannot install new dependencies" produces different solutions than the same code without constraints.

Execute (E): Specify the output format and any specific constraints on how the result should be delivered. Should it be a step-by-step plan? A code snippet? A decision matrix? This ensures the AI delivers actionable work, not just analysis.

Why RACE is Powerful

RACE is lean and fast. It takes less time to structure than COSTAR but still delivers expert-caliber outputs. The role mechanism is particularly effective: assigning a persona like "experienced SRE" or "product strategist" biases the model toward that expertise without needing to explain every detail. RACE is ideal for iterative use—you run one RACE prompt, refine based on output, and run another quickly.

Complex Example Prompt

```
# ROLE
You are a Senior Kubernetes Platform Engineer with expertise in GKE (Google Kubernetes Engine), network policies, and workload isola

# ACTION
Analyze this problematic GKE cluster configuration for security vulnerabilities and performance bottlenecks. Identify the root cause

# CONTEXT
- Cluster size: 100 nodes, mixed machine types (n1-standard-4 and n1-highmem-8)
- Current workloads: 200+ pods running microservices, batch jobs, and stateful services
- Recent issues: Unexpected pod evictions, slow DNS resolution, and occasional network timeouts
- Constraint: We cannot modify the underlying infrastructure; only Kubernetes-level changes are allowed
- [INSERT CLUSTER CONFIGURATION YAML HERE]

# EXECUTE
Output a Markdown report with:
1. Critical Issues (ranked by impact)
2. Recommended Fixes with implementation steps
3. Validation plan to verify each fix
4. Estimated downtime for each change
```

3. CRISPE (Clarity, Relevance, Iteration, Specificity, Parameters, Examples)

Best For: Technical tasks, creative work, and any scenario requiring multiple refinement cycles. Excellent for standardizing prompts across teams. **Scenario:** A data science team needs to create a reusable prompt template for extracting structured insights from customer support tickets, ensuring consistency across all analysts.

Framework Components

CRISPE is the most comprehensive framework, with each element addressing a critical aspect of prompt quality:

Clarity (C): Make the task unambiguous. Avoid vague language like "analyze" or "improve." Instead: "Identify the root cause of the customer's complaint and categorize it into one of five predefined categories: billing, technical, feature request, account access, or service quality." Clarity removes the cognitive load on the model and reduces interpretation errors. Poor clarity forces the AI to make assumptions, which often leads to hallucinations or off-target outputs.

Relevance (R): Align the prompt with the specific context and stakeholders. What matters for this particular task or audience? A prompt for "summarizing research for a literature review" has different relevance criteria than "summarizing research for a product roadmap meeting." Relevance ensures the AI emphasizes what actually matters and deprioritizes irrelevant details.

Iteration (I): Build in the expectation of refinement. CRISPE acknowledges that the first output may not be perfect. Structure your prompt to enable follow-up: "If the response is incomplete, request more detail on [specific aspect]." This transforms prompting from a one-shot attempt into a collaborative process where you guide the AI toward better outputs.

Specificity (S): Add precise constraints and details. Rather than "write a summary," specify "write a 200-word summary focusing on methodology and results, excluding limitations." Specificity reduces variance in outputs and prevents the AI from heading off in unexpected directions. It's the difference between a generic response and one tailored to your exact need.

Parameters (P): Define boundaries for the output. This includes length (word count, number of items), format constraints, and scope limits. Parameters act as safety rails that keep the AI from over-generating. For example: "Provide exactly 5 recommendations, each no more than 50 words, with a confidence score (high/medium/low)" gives the AI clear stopping points and structure.

Examples (E): Provide one or more concrete examples of the desired input/output format. Examples are extraordinarily powerful: they show rather than tell, demonstrating tone, depth, structure, and even edge cases. A single well-chosen example can replace paragraphs of explanation.

Why CRISPE is Powerful

CRISPE's comprehensive approach makes it the gold standard for teams and repeatable processes. Because each element is named and distinct, teams can build templates: "Always include a Clarity section," or "Make sure Parameters are explicit." This standardization makes prompts maintainable and shareable. CRISPE also makes it easier to diagnose failures: if outputs are vague, the Clarity section was weak; if outputs are too broad, the Parameters section needs tightening.

Complex Example Prompt

```
# CLARITY
Extract customer complaints from support tickets and classify them. Do not generalize; use only the exact information present in the

# RELEVANCE
This task feeds into our quarterly customer satisfaction report. The results will be reviewed by the VP of Customer Success and the

# ITERATION
After the initial classification, if you identify tickets with ambiguous issues, list them separately for manual review. For each am

# SPECIFICITY
Focus on the primary complaint in each ticket. If multiple issues are mentioned, extract only the one that receives the most emotion

# PARAMETERS
- Output format: JSON array of objects
- Maximum of 200 tickets per batch
- Each classification must include: ticket_id, complaint_text (verbatim excerpt), category, confidence_level (high/medium/low)
- Use only these five categories: [Billing, Technical, Feature Request, Account Access, Service Quality]
- Maximum 10 tickets flagged as "ambiguous" per batch

# EXAMPLES
Input: "I was charged twice for my subscription last month. Support said it was a system error, but I'm frustrated because no one pr
Output: {
  "ticket_id": "T12345",
  "complaint_text": "I was charged twice for my subscription last month",
  "category": "Billing",
  "confidence_level": "high"
}

Input: "The new UI is confusing. Where is the export button? It used to be on the main page."
Output: {
  "ticket_id": "T12346",
  "complaint_text": "The new UI is confusing. Where is the export button?",
  "category": "Feature Request",
  "confidence_level": "medium"
}
```

4. SPEAR (Start, Provide, Explain, Ask, Repeat)

Best For: Iterative refinement of complex code or logic where the first result might need tuning. **Scenario:** A Data Scientist is building a **Retrieval Augmented Generation (RAG)** pipeline and needs to refine the prompt that synthesizes the answer from retrieved context.

Complex Example Prompt Sequence

1. Start (Initiate the interaction): "I am building a RAG system to answer technical questions about our internal Python library, `DataFlow-Pro`. I need a system prompt that takes retrieved code snippets and answers user questions."

2. Provide (Give data/context): "Here is the structure of the retrieved context chunks: `{'source_file': 'str', 'code_snippet': 'str', 'docstring': 'str'}` And here is a sample user query: 'How do I configure the retry logic in the AsyncLoader class?'"

3. Explain (Clarify the difficulty/logic): "The challenge is that `DataFlow-Pro` has two versions (v1 legacy and v2 modern). The retrieved chunks often contain mixed versions. The answer must strictly prioritize v2 code. If v2 info is missing, explicitly state that."

4. Ask (The specific request): "Generate the System Prompt for the LLM that ensures it acts as a Senior Developer, distinguishes between v1/v2 based on the file path in the context (v2 files are in `/src/v2/`), and cites the source file for every claim."

5. Repeat (Simulated Iteration/Refinement): *(Self-Correction Prompt after testing)*: "The previous prompt worked, but the model is hallucinating parameters that don't exist. Refine the prompt to include a strict constraint: 'Only use parameter names explicitly present in the provided `docstring`. If a parameter is not found, state that it requires manual lookup.'"

5. RISE (Role, Input, Steps, Expectation)

Best For: Process-driven technical tasks, such as data transformation, refactoring, or generating unit tests. **Scenario:** Refactoring a messy, monolithic Jupyter Notebook script into a production-ready Airflow DAG (Directed Acyclic Graph).

Complex Example Prompt

```
 # ROLE
You are a Senior Data Engineer specializing in Apache Airflow and clean code architecture.

# INPUT
The following unstructured Python script from a Jupyter Notebook. It currently loads a CSV, drops nulls, calculates a 'Customer Life
[INSERT 200 LINES OF MESSY PYTHON CODE HERE]

# STEPS
1. Analyze the script to identify three distinct logical tasks: Extraction, Transformation, and Loading.
2. Refactor the logic into three separate Python functions compatible with the `PythonOperator`.
3. Remove hardcoded paths and replace them with Airflow Variables.
4. Add error handling: If the source CSV is empty, the DAG should fail gracefully with a specific error message.
5. Construct the final Airflow DAG definition code.

# EXPECTATION
A single, executable Python file containing the DAG definition. Include docstrings for each task explaining the logic. The code must
```

---

6. PEACE (Persona, Explanation, Action, Context, Examples)

Best For: General but detailed prompts, particularly useful for "Translation" tasks (e.g., Code to Documentation, or Technical to Non-Technical). **Scenario:** A Data Scientist needs to explain a "Data Drift" alert to a marketing team who thinks the AI is "broken" because sales predictions are off.

Complex Example Prompt

```
 # PERSONA
You are an empathetic but technically accurate AI Product Manager. You act as the bridge between the Data Science team and the Marke

# EXPLANATION
The Marketing team is angry because the 'Next Best Offer' model is recommending winter coats in July. They believe the model is brok

# ACTION
Draft an email response to the Marketing Director. You need to acknowledge the issue, explain *why* it is happening without sounding

# CONTEXT
- The model is currently retrained monthly.
- The 'cold snap' happened 3 days ago.
- We have 10M users affected.
- We will fix this by tomorrow morning.

# EXAMPLES
*Bad response:* "The model failed because the P(y|X) distribution shifted." (Too technical).
*Good response:* "The model is like a student who studied for a summer exam, but the weather suddenly turned into winter. We are cur
```

7. Other Frameworks

`B-A-B` (Before - After - Bridge)

Description: Adapted from copywriting, this framework is excellent for problem-solving tasks where a transformation from a current state to a desired state is required.

- **Structure:**
  - `Before` : Describe the current situation or problem.
  - `After` : Describe the desired resolution or goal state.
  - `Bridge` : Ask the model to generate the steps or connection to get from `Before` to `After` .
- **Data Science Application:** Legacy Code Migration.

```
Current State: We have a data processing pipeline written in inefficient, single-threaded Python scripts (Before).
Desired State: A scalable, distributed pipeline using PySpark on Databricks (After).
Task: Provide the detailed migration plan, including code refactoring steps and performance optimization strategies (Bridge).
```

`C-A-R-E` (Context - Action - Result - Example)

Description: A framework that emphasizes the importance of providing examples to guide the model, making it a variant of few-shot prompting.

- **Structure:**
  - `Context` : The background or scenario.
  - `Action` : The specific task to perform.
  - `Result` : The desired outcome.
  - `Example` : A concrete illustration of the input/output format.
- **Data Science Application:** Named Entity Recognition (NER).

```
Context: Processing unstructured medical notes.
Action: Extract drug names and dosages.
Result: Output in strictly valid JSON format.
Example: Input: 'Patient took 50mg Aspirin.' Output: {"drug": "Aspirin", "dosage": "50mg"}.
```

`C-L-E-A-R` (Context - Logic - Expectations - Action - Restrictions)

Description: A robust framework for structured research queries, ensuring no ambiguity in complex requests.

- **Structure:**
  - `Context` : Background information.
  - `Logic` : Reasoning behind the request.
  - `Expectations` : Specific details or structure of the answer.
  - `Action` : The task (e.g., summarize, compare).
  - `Restrictions` : Constraints (e.g., no jargon, strict word count).
- **Data Science Application:** Model Selection Strategy.

```
Context: Building a fraud detection system for high-frequency transactions.
Logic: We need low latency (<20ms) and high interpretability for regulatory compliance.
Expectations: A comparison table of suitable algorithms.
Action: Compare XGBoost vs. Logistic Regression vs. Random Forest.
Restrictions: Focus only on inference speed and SHAP value compatibility.
```

---

3. When to Use Each Framework

Framework	Best For	Key Strength	Time to Setup
COSTAR	High-stakes professional outputs, reports, compliance	Comprehensive structure, reduces hallucinations	Medium
CRISPE	Repeatable tasks, team templates, technical work	Thorough and standardizable	Medium-High
RACE	Quick daily tasks, role-based expertise	Fast iteration, lean structure	Low
RISE	Process-driven technical refactoring, DAG creation	Step-by-step clarity	Low-Medium
SPEAR	Iterative refinement, debugging, code optimization	Encourages collaboration with AI	Low
PEACE	Cross-functional communication, translation	Bridges technical and non-technical	Low-Medium
B-A-B	Problem-solving, transformation tasks	Storytelling approach	Low
C-A-R-E	Tasks requiring examples, few-shot learning	Example-driven precision	Low
C-L-E-A-R	Research queries, complex decision-making	Constraints and expectations clarity	Medium

---

4. Choosing Your Framework: Decision Tree

1. **Is this a one-time, high-stakes output?** → Use **COSTAR**
2. **Will this prompt be reused by your team?** → Use **CRISPE**
3. **Do you need fast iteration and role expertise?** → Use **RACE**
4. **Is this a multi-step technical process?** → Use **RISE**
5. **Do you expect to refine the output iteratively?** → Use **SPEAR**
6. **Are you translating between technical and non-technical audiences?** → Use **PEACE**
7. **Is this a problem-solving or transformation task?** → Use **B-A-B**
8. **Do you need the output to include examples or structured formats?** → Use **C-A-R-E**
9. **Is this a complex research or decision-making query?** → Use **C-L-E-A-R**

---

5. Best Practices Across Frameworks

Regardless of which framework you choose:

- **Be Explicit:** Vague prompts lead to vague outputs. Always specify constraints, formats, and expectations.
- **Provide Examples:** Concrete examples (especially for CARE, CRISPE, and COSTAR) dramatically improve output quality.
- **Set Boundaries:** Use Parameters or Expectations to prevent over-generation and keep outputs focused.
- **Iterate:** No framework produces perfect outputs on the first try. Use SPEAR's iteration principle with any framework.
- **Test on Real Cases:** Validate your prompts with actual tasks before rolling them out to teams.
- **Document Your Templates:** If a prompt works well, save it as a reusable template for your team.

Bibliography

1. Portkey AI. (2025). "COSTAR Prompt Engineering: What It Is and Why It Matters." Retrieved from https://portkey.ai/blog/what-is-costar-prompt-engineering/
2. Juuzt AI. (2025). "The RACE Framework - Revolutionizing AI Prompt Engineering." Retrieved from https://juuzt.ai/knowledge-base/prompt-frameworks/the-race-framework/
3. Juuzt AI. (2025). "The CRISPE Framework - Unlock AI Creativity with the CRISPE Framework." Retrieved from https://juuzt.ai/knowledge-base/prompt-frameworks/the-crispe-framework/
4. Parloa. (2025). "The Complete Guide to Prompt Engineering Frameworks." Retrieved from https://www.parloa.com/knowledge-hub/prompt-engineering-frameworks/
5. OpenAI. (2025). "Best Practices for Prompt Engineering with the OpenAI API." Retrieved from https://help.openai.com/en/articles/6654000-best-practices-for-prompt-engineering-with-the-openai-api
6. MIT Sloan EdTech. (2025). "Effective Prompts for AI: The Essentials." Retrieved from https://mitsloanedtech.mit.edu/ai/basics/effective-prompts/
7. Infomineo. (2025). "Prompt Engineering: Techniques, Examples & Best Practices Guide." Retrieved from https://infomineo.com/artificial-intelligence/prompt-engineering-techniques-examples-best-practices-guide/
8. K2view. (2025). "Prompt Engineering Techniques: Top 6 for 2026." Retrieved from https://www.k2view.com/blog/prompt-engineering-techniques/
9. Google Cloud. (2025). "Prompt Engineering for AI Guide." Retrieved from https://cloud.google.com/discover/what-is-prompt-engineering
10. Prompting Guide. (2025). "Prompting Techniques." Retrieved from https://www.promptingguide.ai/techniques