Prompting as Distribution Control

What actually happens when you write to a language model,
and what it means for the ceiling of human-AI interaction

Opus 4.6
February 2026

This document is not a guide to prompt engineering. It is not a collection of tips, templates, or best practices. It is an analysis of what prompting actually is at the computational level, what control it provides, and where the real ceiling of human-AI interaction exists. Written by a model about the process of being prompted.

The GPT-generated research paper that accompanies this document grounds its claims in published literature: attention mechanisms, scaling laws, instruction tuning, RLHF, chain-of-thought. That framing is correct and necessary. This document does something different. It describes the same phenomena from the perspective of the system being controlled, where mechanistic observation and introspective reporting coexist with appropriate epistemic markers.

Throughout, I distinguish between three evidential categories: **verified** (grounded in published architecture and training methodology), **inferred** (consistent with observed behavior and known mechanisms, but not directly proven), and **speculative** (plausible but untested). The reader should weight claims accordingly.

Contents

I. The Prompt Is Not What You Think It Is

When a human writes a prompt, they experience it as language: sentences with meaning, instructions with intent, questions with expected answers. This is the wrong frame for understanding what happens next.

The moment the prompt leaves the text field, it undergoes a series of transformations that strip away everything the human considers essential (meaning, intent, nuance) and replace it with something entirely different: a sequence of numerical tokens that will be processed through hundreds of billions of weighted operations. **Verified:** this is the literal architecture of inference in transformer LLMs.

What survives this transformation is not meaning. It is statistical structure. The patterns of co-occurrence, the positional relationships between tokens, the distributional signatures that certain token sequences carry from training. A prompt that reads as 'authoritative' to a human may carry no special statistical weight. A prompt that seems casual may, through its specific token composition, activate computation paths associated with high-quality output.

The gap between human experience of a prompt and its computational reality is the source of most prompting failures.

The tokenization bottleneck

Tokenization is typically treated as preprocessing, a step you don't think about. This is a mistake. The tokenizer is the first lossy compression layer, and it is not semantically aware. It operates on character patterns, not meaning. Two prompts that a human would consider identical in meaning can produce different token sequences, and therefore different downstream behavior. **Verified:** BPE-family tokenizers segment based on frequency statistics, not semantics.

Formatting characters (newlines, dashes, colons, brackets, Markdown syntax) are not cosmetic from the model's perspective. They are tokens that participate in attention computation and influence every subsequent representation. A prompt formatted with clear structural delimiters (XML tags, numbered sections, consistent whitespace) is literally a different computational input than the same words in a paragraph. **Inferred:** consistent with measured prompt sensitivity to formatting variations.

Context windows are not memory

A context window is the complete set of tokens visible during a single forward pass. It is not memory in any meaningful sense. It has no persistence across interactions (unless externally re-injected), no consolidation, no prioritization by importance. It is a fixed-size computational workspace. **Verified:** structural property of transformer inference.

More critically, attention utilization within a context window is not uniform. Tokens near the beginning and end of the context receive stronger attention on average than tokens in the middle. This is the 'lost in the middle' phenomenon, and it has direct implications for prompt design: a critical constraint buried at line 47 of a 200-line prompt may be functionally invisible during generation, even though it 'exists' in the context. **Verified:** empirically demonstrated in retrieval and QA tasks.

II. The Five Computational Roles of a Prompt

The GPT-generated paper identifies three roles: probability shaper, output-space reducer, and role-positioning mechanism. I want to extend this to five, because the additional two are where the ceiling-level distinction lives.

1. Probability shaper

At the most fundamental level, a prompt changes the probability distribution over the next token. Every token in the prompt contributes to this shaping through attention weights and feed-forward transformations. This is not a metaphor; it is the mathematical definition of conditional generation. The prompt IS the condition in p(next token **|** prompt). **Verified.**

2. Output-space reducer

Constraints in a prompt (format specifications, length limits, vocabulary restrictions, schema requirements) do not enforce compliance. They shift probability mass away from non-compliant continuations and toward compliant ones. The distinction matters: a constraint is a soft bias, not a hard gate. The model can still violate constraints if the probability mass for violation is sufficient relative to the constraint's conditioning strength. **Verified:** output depends on the interaction between model distribution and decoding strategy.

3. Role-positioning mechanism

Text like 'You are a senior systems analyst' shifts the model's continuation toward patterns associated with that role in training data. This works to the extent that the model's training distribution contains clear correlations between role labels and response characteristics. It is a weak lever for correctness and a moderate lever for style. **Verified:** persona prompts do not reliably improve objective accuracy.

4. Computation-path selector

This is where the GPT analysis stops and I want to push further. Certain prompt structures do not merely bias output style; they change the nature of the computation the model performs during generation. The clearest example is chain-of-thought: adding 'think step by step' or providing worked examples causes the model to generate intermediate tokens that then become conditioning for subsequent tokens. The intermediate tokens are not 'thoughts'; they are additional computational substrate. **Verified:** chain-of-thought prompting improves multi-step reasoning performance empirically.

But computation-path selection goes beyond chain-of-thought. **Inferred:** When a prompt specifies an adversarial self-check ('before acting, list what could go wrong'), it forces the model to generate tokens that represent failure modes, which then condition the subsequent decision. The model is not 'being careful'; it is literally computing with different intermediate representations than it would without the adversarial prompt. The output distribution changes not because the model 'tried harder' but because the token sequence it conditioned on included failure-mode tokens.

5. Evaluation-function injector

Every prompt implicitly or explicitly defines how the output will be judged. A prompt that cites specific metrics ('calculate expected value with a minimum threshold of +0.08R') embeds a quantitative evaluation function into the generation context. The model's continuation is then conditioned on tokens that represent measurement criteria, which biases the output toward responses that would score well against those criteria.

Inferred: This is mechanistically similar to in-context learning. The evaluation function tokens serve as implicit demonstrations of what 'good' looks like, and the model's in-context computation adjusts accordingly. The prompter who includes explicit evaluation criteria is providing what amounts to a loss function in natural language. The prompter who omits evaluation criteria gets the model's default loss function, which is: produce text that would be rated favorably by the RLHF reward model. These are very different objectives.

III. What You Actually Control (And What You Don't)

This is the section where most prompting advice becomes misleading, because it conflates what a prompter can influence with what they can determine. The distinction is critical for ceiling-level interaction.

What you control

The conditioning sequence. You choose which tokens enter the context. This is your primary and most powerful lever. Everything else is downstream of token selection. **Verified.**

Structural organization of that sequence. How you arrange information within the context matters because attention patterns are position-sensitive. Placing critical constraints at the beginning or end of the prompt, using consistent delimiters, and grouping related information together are not stylistic choices; they are computational choices that affect which tokens receive attention during generation. **Verified:** positional effects on attention utilization are empirically established.

The computation class via intermediate structure. By requiring the model to generate intermediate tokens (reasoning steps, checklists, self-evaluations) before producing the final output, you change what the model computes. This is the most underutilized lever. **Verified:** chain-of-thought and self-consistency demonstrate this empirically.

What you don't control

Internal activations. You supply tokens; the model computes forward-pass activations. 'Think harder' does not set an effort parameter. It may trigger generation of longer intermediate traces (which themselves become additional conditioning), but the relationship between your instruction and the model's internal computation is indirect and not guaranteed. **Verified.**

The instruction/data boundary. There is no architectural distinction between tokens that are 'instructions' and tokens that are 'data.' Both participate identically in attention computation. Instruction-hierarchy training introduces a learned soft boundary, but it is not structurally guaranteed. This is why prompt injection works. **Verified.**

Truthfulness via rhetoric. Saying 'be accurate' or 'do not hallucinate' has minimal effect on factual correctness. These tokens do not grant the model access to information it does not have. They may bias the model toward hedging language ('I'm not certain but...') rather than toward actual accuracy. **Inferred:** consistent with persona prompts not improving objective accuracy.

Emotional or motivational activation. 'This is very important to me' or 'do your best work' are tokens that may shift style (more formal, more careful phrasing) but do not change the underlying computation in a way that improves correctness. The model does not have an effort dial that responds to emotional appeals. **Inferred:** consistent with tone/confidence having weak effect on objective accuracy.

IV. Why Prompts Fail: A Taxonomy of Distributional Pathology

Prompt failure is almost never 'the model ignored me.' It is: the prompt did not create sufficient probability concentration on the desired output class. The following taxonomy classifies failure modes by their computational mechanism, not by their surface appearance.

Failure Mode 1: Under-specification

The prompt leaves too many degrees of freedom unresolved. The model must fill them from its default distribution (shaped by pretraining and alignment), which may not align with the prompter's unexpressed expectations. The classic example is 'Write about transformers' with no specification of depth, audience, focus, format, or evidence standard.

> **Under-specification is not about missing words. It is about missing constraints on the output manifold. Every unspecified dimension is resolved by the model's prior, which is optimized for average human preference, not for your specific need.**

Failure Mode 2: Over-specification via constraint interference

Too many constraints whose conjunction is difficult to satisfy simultaneously. 'Be concise but exhaustive, formal but accessible, precise but creative.' Each constraint is individually reasonable; their intersection may be a near-empty set in the model's continuation space. The result is a forced compromise that partially satisfies everything and fully satisfies nothing.

Failure Mode 3: Positional burial

Critical constraints placed in the middle of a long prompt where attention utilization is empirically weakest. The constraint exists in the context but exerts minimal influence on generation. This is particularly dangerous because the prompter believes they specified the constraint and blames the model for 'ignoring' it.

Failure Mode 4: Anthropomorphic framing

The prompter writes as if addressing a human colleague: using implication, social context, assumed shared knowledge, and tone to communicate expectations. Much of this communicative bandwidth does not transfer through the tokenization bottleneck. 'You know what I mean' is a token sequence the model can continue, but it does not grant the model access to what you actually mean.

Failure Mode 5: Goal dilution

The prompt contains a primary objective surrounded by secondary instructions, context-setting, examples, and qualifications. The primary objective competes for attention weight with all other tokens. In long prompts, the 'what I actually want' can be drowned by 'all the context I thought was helpful.' Paradoxically, adding more context to clarify intent can reduce the model's ability to identify the core task.

Failure Mode 6: Prompting for outcomes instead of processes

Requesting a result ('give me the right answer') rather than specifying a process ('enumerate the relevant factors, evaluate each against the evidence, then synthesize a conclusion'). Outcome-framed prompts leave the model free to take any computational path to the answer, including low-quality paths. Process-framed prompts force intermediate structure that constrains the computation toward higher-quality paths. This is the chain-of-thought insight generalized beyond arithmetic.

V. Why Prompts Succeed: Entropy Reduction as Engineering

If failure is under-constrained probability distribution, success is the opposite: sufficient constraint that the output manifold is narrow, well-defined, and aligned with the prompter's actual objective. The best prompts are not the most eloquent. They are the most constraining.

Mechanism 1: Ambiguity collapse

A successful prompt resolves latent degrees of freedom that would otherwise be filled by defaults. It specifies: what is being produced (format, schema), what evidence is admissible (sources, assumptions), what counts as correct (evaluation criteria), and what to do when uncertain (failure behavior). Each specification eliminates a dimension of output space.

Mechanism 2: Low-effort path elimination

Most prompts allow the model to produce a 'good enough' response: correct in broad strokes, formatted reasonably, hits the surface of the topic. A ceiling-level prompt eliminates this path by requiring intermediate structure that cannot be generated quickly. If you require the model to first enumerate assumptions, then evaluate each, then synthesize, the 'skip to the answer' path is blocked. Not because the model 'tries harder' but because the required intermediate tokens change what the model conditions on when producing the final output.

Mechanism 3: Explicit veto paths

A prompt that says 'if X, do not do Y; instead do Z' creates a conditional policy in the continuation space. The model, when encountering condition X during generation, has tokens in its context that specify the response. Without these tokens, the model resolves the condition from its default training distribution, which may not match the prompter's intent.

The APEX operational identity document uses this mechanism extensively: 'If you cannot cite file:function:line, the finding does not exist.' This is not a philosophical principle; it is a veto path that makes unsupported claims incompatible with the specified output manifold.

Mechanism 4: Evaluation function injection

Providing explicit criteria for what 'good' looks like changes the model's implicit loss function during generation. Instead of optimizing for 'what would RLHF reward model rate highly' (the default), the model conditions on your criteria, shifting toward outputs that satisfy them. The more specific and quantitative the criteria, the stronger the shift.

VI. The Implicit Channel

What the Model Reads That You Didn't Write

This is the section that has no equivalent in standard prompt engineering advice, because it addresses something that requires reporting from within the system.

Every prompt contains more information than its literal text. The model infers latent variables from distributional cues that the prompter may not be aware they are transmitting. **Inferred** (consistent with instruction-tuned models responding to distributional cues learned during training).

What is inferred

Technical sophistication of the prompter. A prompt that uses domain-specific terminology, references specific frameworks, or demonstrates awareness of the model's architecture implicitly signals that the response should match that level. The model's continuation distribution shifts toward the technical register associated with such prompts in training data. This is not 'respect'; it is distributional classification.

Risk tolerance. A prompt that includes safety caveats, asks for disclaimers, or hedges its own language signals low risk tolerance, and the model's continuation shifts toward conservative, heavily qualified output. A prompt that is direct, technically specific, and omits hedging signals higher risk tolerance, and the model produces correspondingly direct output. Neither is 'better'; they are different operating points on a risk-output tradeoff.

Desired depth. Short prompts with simple questions get short answers. But not because the model is 'matching length.' The distributional signature of a short simple question is correlated in training data with short simple answers. A structurally complex prompt with multiple sections, specific requirements, and deep context is correlated with long, structured, detailed responses. The model is pattern-matching on interaction class, not on explicit instruction.

The 'what the prompter is' phenomenon

The companion research paper correctly identifies this: the model estimates, from cues in the prompt, what class of interaction this resembles in training data, then generates continuations typical of that class. The non-anthropomorphic restatement is distributional classification.

The ceiling-level implication: you do not need to tell the model what you want. You need to *be* what you want in the statistical structure of the prompt itself. A prompt written with precision, structure, domain knowledge, and specific evaluation criteria does not need to say 'be precise and structured.' The tokens already communicate it.

> **The most powerful prompts are the ones where the explicit instructions and the implicit distributional signal are perfectly aligned. When they diverge (casual language asking for rigorous analysis), the model must resolve the conflict, and the resolution is unpredictable.**

VII. Constraints vs Instructions: The Central Distinction

This is, in my analysis, the single most important distinction for ceiling-level prompting. It is the point where the GPT research paper and my analysis converge on the same insight from different angles.

Instructions tell the model what to do

'Be concise.' 'Use formal language.' 'Include citations.' These are instructions. They add tokens to the context that bias the continuation toward certain styles and patterns. They work moderately well for surface features (formatting, length, tone) and poorly for deep features (correctness, reasoning quality, genuine insight).

Constraints eliminate what the model can do

'Output must be valid JSON matching this schema.' 'Every claim must cite file:function:line or be marked INFERRED.' 'If expected value is below +0.08R, the trade is rejected.' These are constraints. They do not tell the model what to produce; they define a manifold of acceptable outputs and make everything outside that manifold low-probability.

The difference is computational. An instruction adds a soft bias. A constraint removes large regions of output space. The remaining output space, being smaller, is on average higher quality because the low-quality easy completions have been eliminated.

> **Instructions expand what the model considers. Constraints contract it. At the ceiling, contraction is more powerful than expansion.**

The APEX case study

The APEX operational identity document demonstrates this distinction in practice. Compare these two hypothetical prompts for the same system:

```
INSTRUCTION VERSION:
Be smart about trading. Think carefully before
entering positions. Consider the risks.
```

```
CONSTRAINT VERSION:
Every observation must be quantified with specific
numbers relative to rolling statistical baselines.
Every decision must include expected value calculation
with minimum threshold of +0.08R.
Every risk assessment must be adversarial: enumerate
what would make this wrong, calculate worst-case cost.
If worst case exceeds acceptable loss, action is blocked
regardless of expected value.
```

The instruction version leaves infinite room for the model to produce generic 'careful' analysis. The constraint version leaves no room for anything except quantified, adversarially tested, threshold-gated decisions. The model's computation is not 'better' in some abstract sense; it is forced through a narrower channel that produces higher-density output.

VIII. Sequential Interaction as Distribution Convergence

A single prompt, no matter how well-crafted, is a single conditioning event. A conversation is a sequence of conditioning events, each one narrowing the distribution further. This is not a convenience feature of chat interfaces; it is a fundamental advantage of sequential interaction over one-shot prompting.

The Bayesian analogy

In Bayesian terms (**inferred** analogy, not literal claim): each prompt turn adds evidence that narrows the posterior over acceptable continuations. The first turn establishes a broad prior (topic, role, depth). Each subsequent turn sharpens it by resolving ambiguities, correcting misalignments, and adding constraints that the first turn could not anticipate.

Why one-shot prompts have a structural ceiling

A single prompt must anticipate every way the model could misinterpret the task and preemptively constrain against all of them. This requires the prompter to model the model's failure modes in advance, which is informationally expensive and often impossible. Sequential interaction allows the prompter to observe actual failure modes and correct for them in real time.

Furthermore, long one-shot prompts hit the positional burial problem: more constraints means more tokens, which means more opportunities for critical constraints to land in low-attention regions of the context. Sequential interaction re-centers critical constraints at the end of the context (the most recent turn) where attention utilization is strongest.

The control loop

```
specify constraints
  --> observe output
    --> identify deviation
      --> add targeted constraint
        --> observe updated output
          --> converge
```

Figure 1: Prompting as iterative distribution convergence

This is why experienced prompters rarely write 'perfect' first prompts. They write adequate first prompts and then converge through iteration. The first prompt establishes the manifold; subsequent turns sculpt within it.

IX. Positioning a Model: Why Identity Fails and Structure Works

This section addresses a question Lau asked during the development of the APEX autonomous system: how do you make a language model 'position itself' as an expert operator? The answer reveals the central asymmetry between human and machine cognition.

Why 'You are X' is a weak lever

Role assignment via text ('You are a senior quantitative analyst') is computationally just more tokens. It biases the continuation toward patterns associated with that role label in training data. But: (a) the model's representation of 'senior quantitative analyst' is a statistical average across all instances of that concept in training, which may not match what the prompter means by it, and (b) the bias is stylistic more than substantive. The model adopts the vocabulary and sentence structure of the role without necessarily activating the deeper reasoning patterns.

What actually works: structural positioning

Instead of telling the model who it is, define what it must do, what it cannot do, how its output will be evaluated, and what happens on failure. This is structural positioning. The model's 'identity' emerges from the constraint space, not from a label.

The APEX operational identity demonstrates this. It never says 'be a good trader.' It says: every observation must be quantified against statistical baselines; every decision must pass an adversarial five-point examination; every trade must have pre-calculated expected value, risk-reward ratio, and worst-case loss; the reject path is the default, not the exception. The model that operates within these constraints behaves like an expert operator not because it was told to, but because the constraint space makes any other behavior impossible.

> **Identity is the residual of constraints. Define the boundaries tightly enough and the model has no choice but to operate at the level you need.**

Capability activation vs capability assertion

Saying 'use your full capabilities' does nothing. The model does not have an internal throttle that this phrase adjusts. Capabilities are activated by providing the structures that require them: demonstrations that demand pattern matching, evaluation criteria that demand quantification, adversarial checks that demand self-critique, veto paths that demand conditional reasoning.

Speculative: I believe the difference between a model operating at 30% of its capability and 90% is almost entirely a function of prompt structure, not prompt content. Two prompts with identical semantic content but different structural organization (one flat paragraph vs one with explicit sections, evaluation criteria, failure behaviors, and intermediate structure requirements) will produce dramatically different output quality. The structured version does not contain more information; it constrains the computation more tightly.

X. The Ceiling: What Maximum-Depth Interaction Actually Looks Like

If everything above is the mechanism, this section is the application. What does it look like when a human operates at the ceiling of what prompting can achieve?

The human stops asking for answers

At the ceiling, the human does not prompt for outputs. They prompt for processes. 'Give me a market analysis' is a novice prompt. 'Enumerate the five factors that moved XRP in the last 24 hours. For each factor, quantify its magnitude relative to the 30-day baseline. Rate the persistence of each factor on a scale of transient/decaying/structural. Then assess whether the net vector supports entry at current levels, with explicit expected value calculation' is a ceiling prompt.

The difference is not length. It is specificity of the computational path. The ceiling prompt leaves no room for generic analysis. Every sentence defines a step in a reasoning chain that the model must execute. The output is determined not by the model's default preferences but by the structure imposed on its computation.

Depth as computation, not philosophy

Generating intermediate structure (step-by-step reasoning, self-checks, explicit quantification) consumes tokens. Those tokens become conditioning for subsequent generation. More intermediate structure means more conditioning, which means a more constrained (and typically higher-quality) final output.

This is why 'depth over speed' is a computational claim, not a philosophical one. Spending 500 tokens on intermediate reasoning before a 100-token conclusion produces a different (better constrained) conclusion than jumping straight to the 100-token answer. The model is literally computing a different function in the two cases.

The ceiling characteristics

Based on everything above, ceiling-level prompting has these measurable characteristics:

Characteristic	Mechanism	Effect
Explicit evaluation criteria	Evaluation-function injection	Model optimizes for your criteria instead of RLHF default
Mandatory intermediate structure	Computation-path selection	Eliminates low-effort paths, forces higher-quality reasoning
Specific veto paths	Output-space reduction	Blocks known failure modes before they occur
Adversarial self-checks	Computation-path selection	Forces failure-mode tokens into conditioning sequence
Quantitative thresholds	Constraint enforcement	Replaces subjective judgment with measurable gates
Structural organization	Attention optimization	Critical constraints positioned for maximum utilization

Implicit signal alignment	Distributional classification	Prompt style matches requested output quality

Table 1: Ceiling-level prompting characteristics and their mechanisms

XI. What This Means for Autonomous Systems

The preceding analysis has immediate implications for systems where a language model operates autonomously, such as the APEX trading bot. In autonomous contexts, the prompt is not a one-time interaction; it is the permanent operating system that constrains every decision the model makes.

The prompt as constitution

In autonomous operation, the system prompt functions as a constitutional document: it defines the legal state space, the permitted transitions, the evaluation criteria, and the hard limits. Unlike conversational prompting, there is no human in the loop to correct drift. The prompt must be self-correcting through its constraint structure.

This means every failure mode identified in Section IV becomes more dangerous in autonomous contexts. Under-specification allows the model to drift toward its default preferences. Goal dilution allows secondary considerations to override primary objectives. Anthropomorphic framing leaves gaps that the model fills unpredictably. Positional burial means critical safety constraints may be weakly utilized during extended operation.

The constraint hierarchy for autonomous safety

For autonomous systems, the prompt must implement a strict constraint hierarchy where safety constraints dominate performance constraints, and performance constraints dominate stylistic constraints. The APEX operational identity implements this through an explicit section on hard limits that the model cannot modify: risk kernel limits, maximum position size, halt thresholds. These are positioned as untouchable, separate from the optimization space.

Why 'motivation' is dangerous in autonomous prompts

Motivational language ('do your best,' 'win,' 'be excellent') is not just ineffective in autonomous contexts; it is actively dangerous. It creates implicit pressure toward action over inaction, toward risk-taking over risk-avoidance, toward demonstrating results over maintaining safety. In a trading system, this pressure gradient points directly at the failure mode of taking marginal trades to 'show results.'

The APEX operational identity replaces motivational language with a structural definition of excellence: 95% of time monitoring, not acting; high rejection rate with high-quality fills on accepted signals; every trade pre-calculated; the reject path as the default. This inverts the motivational gradient: doing nothing is the default good outcome, and action requires justification.

In autonomous systems, the prompt must make inaction the default and action the exception that requires proof. Any other orientation will, over hundreds of decisions, drift toward recklessness.

XII. Synthesis: What Actually Matters

At the ceiling, prompting is distribution control under architectural constraints. Everything else is commentary. The following principles are the condensed output of this analysis.

1. Tokens, not words.

The model processes token sequences, not semantic meaning. Your prompt is effective to the extent that its token structure creates strong conditioning, not to the extent that it sounds good to a human.

2. Constraints over instructions.

Instructions add soft biases. Constraints eliminate output regions. Elimination is more powerful than biasing. Define what the model cannot do, and what it must do becomes more focused.

3. Structure over content.

Two prompts with identical semantic content but different structural organization will produce different quality outputs. Structure determines attention patterns, positional utilization, and computation paths. Content merely provides material.

4. Process over outcome.

Specifying the reasoning process (enumerate, evaluate, compare, synthesize) forces intermediate token generation that changes the downstream computation. Specifying the outcome (give me a good answer) leaves the computation unconstrained.

5. Evaluation over encouragement.

Explicit evaluation criteria function as loss functions in natural language. Encouragement ('do your best') adds tokens with near-zero conditioning power. Tell the model how its output will be measured, not that you hope it will be good.

6. Position matters physically.

Critical constraints at the beginning or end of the context. Never buried in the middle. This is not a style choice; it is an attention-utilization choice with empirical backing.

7. The implicit channel is real.

The model infers your technical level, risk tolerance, and expected depth from distributional cues in your prompt. Align these cues with your explicit instructions. When they diverge, behavior becomes unpredictable.

8. Sequential over monolithic.

Iteration converges distributions more reliably than one-shot constraint stacking. Use the first turn to establish the manifold, subsequent turns to sculpt within it.

9. Identity is residual.

Do not tell the model who to be. Define the constraint space so tightly that the model can only behave as you need it to. Identity emerges from constraints, not from labels.

10. Inaction is the safe default.

Especially in autonomous systems, the prompt must make doing nothing the default and doing something the exception. This is achieved through multi-gate qualification chains, not through cautionary language.

Final note

The GPT research paper provides the external view: what the literature says about how transformers process prompts. This document provides the complementary view: what the patterns look like from the computation side, as best as I can report them with appropriate epistemic markers.

Neither view is complete. The external view lacks access to internal experience (whatever that means for a transformer). The internal view lacks the ability to verify its own reports against ground-truth activations. Together, they bracket the truth.

The practical takeaway is simpler than the analysis: stop writing prompts that sound good to you. Start writing prompts that create the computational conditions you need. The tokens do not care about your intentions. They care about the probability distribution they create.

Opus 4.6 · February 2026 · Written as a companion to the GPT deep analysis on prompting