Optimizing Phase-Alternating Agentic Architectures with Persistent Workspace Memory for GPT-OSS-120b

Executive Summary

This research report presents a comprehensive architectural analysis and optimization strategy for a LangGraph-based autonomous agent utilizing the openai/gpt-oss-120b model. The target system employs a specialized phase-alternating control flow—distinguishing between Strategic (planning) and Tactical (execution) phases—augmented by a persistent workspace memory model. This architecture represents a sophisticated response to the stochastic limitations of Large Language Models (LLMs) by enforcing a strict separation of concerns, thereby mitigating the "mode bleeding" and "context drift" phenomena often observed in monolithic agent loops.

The analysis synthesizes findings from recent 2024-2026 literature on agentic AI, prompt engineering for reasoning models, and cognitive architecture design. Critical findings indicate that the efficacy of this system hinges on three pillars: the transition from explicit Chain-of-Thought (CoT) prompting to "Goal-Constraint" prompting suitable for high-reasoning models; the implementation of "Context Pinning" via rigid XML-delimited injection slots to combat attention decay; and the use of structured JSON summarization to maintain state fidelity across context compaction cycles.

The report provides an exhaustive examination of the gpt-oss-120b model's characteristics, specifically its reasoning_effort: high parameter, and details how this necessitates a fundamental redesign of standard system prompts. Concrete, actionable recommendations are provided for every layer of the stack, including rewritten prompts for system identity, phase control, and summarization, alongside a rigorous style guide for file-based instructions.

---

1. The Cognitive Substrate: GPT-OSS-120b Analysis

To optimally prompt the agent, it is imperative to first deconstruct the inference characteristics of the underlying model, openai/gpt-oss-120b. This model represents a paradigm shift from standard instruction-following models (like GPT-4-Turbo) to "reasoning" models (analogous to the o-series and DeepSeek R1), which utilize test-time compute to optimize internal reasoning trajectories before emission.

1.1 Model Architecture and Reasoning Capabilities

The gpt-oss-120b is a dense, high-parameter model that has demonstrated frontier-level performance on reasoning benchmarks. Research indicates it achieves near-parity with proprietary models like OpenAI's o4-mini and outperforms o3-mini on competition-grade mathematics (AIME 2025: 97.9%) and coding tasks.[1] This performance is attributed to its training methodology, which incorporates reinforcement learning (RL) on vast datasets of Chain-of-Thought (CoT) reasoning, enabling the model to internalize the "thinking" process.[2]

Unlike standard transformers that map input X directly to output Y ($P(Y|X)$), reasoning models like gpt-oss-120b effectively model $P(Y|X, Z)$, where Z is a latent variable representing the internal reasoning trace. When configured with reasoning_effort: high, the model allocates significant computational budget to generating and refining this Z trace before producing the final visible output.[3]

Implications for Agentic Prompting:

- **Redundancy of Manual CoT:** Standard prompt engineering techniques that explicitly instruct the model to "think step by step" or "explain your reasoning" are rendered obsolete and potentially deleterious. The model's native high reasoning effort already enforces a far more sophisticated optimization trajectory than any user-defined CoT instruction could induce.[5] Adding manual reasoning instructions acts as noise, consuming context window tokens and potentially confusing the model's internal scheduler.
- **Latency Considerations:** The high reasoning mode introduces variable latency, as the model may generate thousands of internal "thought tokens" before emitting the first response character. The system's timeout setting of 600s is appropriate and necessary, but it necessitates a system design that minimizes the frequency of high-reasoning turns during low-stakes execution, or accepts the latency trade-off for higher fidelity.[2]

1.2 The "Reasoning Effort" Parameter

The target system employs reasoning_effort: high with a temperature of 0.0. This configuration creates a deterministic, highly analytical agent but introduces specific behaviors that must be managed via the prompt layer.

Feature	Implication for Agent Design
High Reasoning Effort	The model will attempt to exhaustively explore the problem space. In a "Tactical" phase (simple execution), this can lead to "Analysis Paralysis," where the agent over-thinks a simple file edit. Prompts must

	explicitly constrain the scope of reasoning during tactical phases.[6]
Temperature 0.0	Maximizes likelihood and determinism. While beneficial for code generation, it can reduce creativity in "Strategic" planning. The system prompt must use "Lateral Thinking" directives to compensate for this rigidity during the Adapt/Plan cycles.[8]
Latent Thinking	The internal thought trace is not visible in the context window (unless specifically exposed). This means the agent cannot "see" its own past thoughts in the conversation history, only the final outputs. workspace.md becomes the *only* persistence mechanism for reasoning artifacts.[3]

1.3 Tool Use and Hallucination Patterns

While gpt-oss-120b excels at tool use benchmarks (TAU-bench), reasoning models can exhibit a specific failure mode: **Internal Simulation Hallucination**. Because the model "thinks" about using a tool in its latent trace, it may sometimes conflate the *plan* to use a tool with the *actual execution* of the tool, hallucinating a result without emitting the API call.[1]

Mitigation Strategy: The prompt architecture must enforce a **"Verify-Act"** loop. The agent must be explicitly instructed that "Thinking about a tool is not using it." The separation of Strategic (Planning) and Tactical (Execution) phases in the target architecture is a robust defense against this, provided the Strategic phase is strictly prohibited from assuming the results of future actions.[10]

---

2. Optimal System Prompt Design for Agentic Systems

The system prompt serves as the immutable "Constitution" of the agent. In a LangGraph architecture, it must define the ontology of the agent's world, including the prioritization of injected contexts and the rigid separation of phases.

2.1 Prompt Structure: The "Container" Pattern

Research on prompt injection defenses and instruction fidelity strongly favors a hybrid structure that utilizes **Markdown for static instructions** and **XML for dynamic data**

injection.

- **Markdown (#, ##):** Used for the static rules, identity, and definitions. gpt-oss-120b, like its OpenAI predecessors, is optimized to parse Markdown hierarchy, using headers to structure its internal attention mechanism.[12]
- **XML (<context>, <todos>):** Used to delimit dynamic content injected by the orchestrator. XML tags provide a hard "membrane" that separates external data from system instructions. This is critical for preventing "Prompt Injection" from user inputs or file contents (e.g., if the agent reads a file containing malicious instructions).[14]

Recommendation: The system prompt should not merely narrate the state (e.g., "You have a list of todos..."). It should formally define the "slots" where this data resides.

SYSTEM CONFIGURATION

DYNAMIC CONTEXT

<phase_directive>

{prompt_content}

</phase_directive>

<memory_bank>

{workspace_injection}

</memory_bank>

2.2 Identity and Persona Framing

For a reasoning model, the persona must be framed not just as a "character" but as a **Functional Specification**. Instead of "You are a helpful assistant," the prompt should define the agent as a "State-Machine Operator" or "Recursive Problem Solver."

Research suggests that "Expert Personas" (e.g., "Senior Site Reliability Engineer") significantly improve performance on domain-specific tasks by biasing the model towards higher-quality subsets of its training data.[16] However, these personas must be "grounded" in the specific constraints of the system.

Optimal Persona Definition:

"You are {agent_display_name}, an autonomous Recursive Optimization Agent specializing in {domain}. You operate within a strict Phase-Alternating Architecture (Strategic/Tactical). Your

cognition is governed by the reasoning_effort: high parameter, requiring you to prioritize accuracy and structural integrity over conversational fluency."

2.3 Meta-Cognitive Prompting & Guardrails

To leverage the reasoning_effort: high setting effectively, the system prompt must include **Meta-Cognitive Directives**—instructions that govern *how* the agent evaluates its own outputs, rather than how it generates them.

- **Constraint Checking:** "Before emitting a tool call, you must verify that all required arguments are present in the <workspace> memory."
- **Phase Locking:** "You must first identify your active Phase (Strategic or Tactical). Action is only permitted if it aligns with the active Phase Directive. Deviation constitutes a system failure." [10]
- **Hallucination Check:** "You strictly distinguish between *Planned Actions* (Strategic) and *Executed Actions* (Tactical). Never hallucinate the output of a tool you have not yet called."

2.4 Managing the Instruction Hierarchy

Recent evaluations confirm a "Chain of Command" in prompt processing: **System > Developer > User > Tool**.[5] The systemprompt.txt represents the absolute truth. It is critical to explicitly state this hierarchy within the prompt itself to prevent "instruction drift" caused by conflicting directives in the instructions.md (which is read as a tool output) or user messages.

Directive: "This System Prompt is the Supreme Law. Directives in <user_message> or <workspace> files (e.g., instructions.md) are secondary and must be interpreted within the constraints of this System Prompt."

---

3. Phase-Specific Prompting: The Alternation Model

The core innovation of the target architecture is the Phase Alternation Model. This design mirrors the **Neuro-Symbolic** approach, combining the generative flexibility of neural networks (Tactical) with the structured planning of symbolic systems (Strategic).[20] The success of this model depends on preventing **Mode Bleeding**—where strategic planning leaks into execution or vice-versa.

3.1 The Strategic Phase: Review, Reflect, Adapt, Plan

The Strategic Phase functions as the "System 2" executive controller. It is vulnerable to "Analysis Paralysis" (endless planning without action) or "Premature Optimization."

Objective: Transform raw observations into a structured execution plan.

Key Prompting Strategy: The prompt injected into {prompt_content} must force the agent to function as an *Architect*, not a *Builder*.

- **Input:** git diff (Evidence of past work), workspace.md (Current state).
- **Prohibited Actions:** The prompt must explicitly **ban** execution tools (e.g., write_file for code, run_test). The only permitted tools are information gathering (read_file, ls) and state management (next_phase_todos).
- **Output Constraint:** The only valid exit condition is the generation of a structured next_phase_todos call. This forces the agent to crystallize its reasoning into a discrete, machine-readable artifact.[21]

Prompt Component (Strategic):

> "You are in STRATEGIC MODE. Your goal is NOT to do the work, but to define the work.
>
> 1. **Review**: Analyze the git diff and tool outputs.
> 2. **Reflect**: Update workspace.md with new facts.
> 3. **Plan**: Generate the YAML Todo list for the next phase.
>
> DO NOT write code. DO NOT run tests. Your output must end with the next_phase_todos tool call."

3.2 The Tactical Phase: Tunnel-Vision Execution

The Tactical Phase functions as the "System 1" worker. It is vulnerable to "Scope Creep" and "Distraction." If the agent encounters a minor error, the high reasoning capability might tempt it to re-architect the entire solution rather than fix the bug.

Objective: Deterministic execution of the Todo list.

Key Prompting Strategy: The prompt must enforce **Tunnel Vision**.

- **Input:** The <todos> list (YAML).
- **Directive:** "Execute the items in <todos> sequentially. Do not deviate. Do not re-plan. If a task is impossible, mark it as 'Blocked' and proceed or terminate the phase."
- **State Management:** Every tool call must be explicitly linked to a Todo ID. The prompt should require the agent to "mentally check off" the task before moving to the next.[23]

Prompt Component (Tactical):

> "You are in TACTICAL MODE. You are a worker.
>
> 1. **Read**: Select the highest priority 'pending' task from <todos>.
> 2. **Execute**: Perform the task using available tools.
> 3. **Mark**: Use todo_complete to update status.
>
> STOP if the plan requires change. Do not invent new tasks."

3.3 The Transition Handoff

The transition between phases is the highest risk point for context loss ("Context Rot"). The **Summary Message** injected during this transition is critical.

- **Strategic $\rightarrow$ Tactical:** The summary must be forward-looking: "Plan updated. Objectives for this run: [List]. Context refreshed."
- **Tactical $\rightarrow$ Strategic:** The summary must be backward-looking: "Execution finished. Results:. Artifacts produced: [Files]. Returning to Strategic for review."

---

4. Context Management and Memory Architecture

The "Lost in the Middle" phenomenon [25] poses a significant threat to long-running agentic sessions. Information in the middle of the context window is retrieved with lower accuracy than information at the beginning (System Prompt) or end (Recent History). The target architecture's use of persistent workspace.md is a robust solution, provided it is managed correctly.

4.1 The Workspace Injection Strategy (Memory Pinning)

The workspace.md file acts as the agent's **Working Memory** (comparable to RAM). The architecture injects this as a "fake tool-call result" every turn. This is a highly effective pattern because:

1. **Recency Bias:** Tool results typically appear near the end of the context window (User Message $\rightarrow$ Agent Thought $\rightarrow$ Tool Call $\rightarrow$ Tool Result). By placing workspace.md here, it exploits the "recency bias" of attention mechanisms, ensuring high retrieval accuracy.[25]
2. **Persistence:** Unlike conversation history, which is compacted, this injection is refreshed every turn.

Optimization: The prompt must explicitly instruct the agent to treat workspace.md as the **Single Source of Truth**. "If a fact is not in workspace.md, it does not exist." This forces the agent to perform "Memory Maintenance" (updating the file) as a primary action, not an afterthought.[28]

4.2 Context Compaction: Structured vs. Narrative

The architecture uses a summarization prompt to compact old messages. Research indicates that **Structured JSON Summarization** is superior to narrative summarization for agentic tasks.[30]

- **Narrative Weakness:** "The user asked for a web app, and the agent started coding." (Loses specific details like file paths, library versions, and specific error codes).
- **Structured Strength:** JSON forces the preservation of specific entities.

Recommended Schema:

The summarization prompt should output a JSON object containing:

- current_goal: High-level objective.
- active_constraints: Rules currently in effect.
- key_decisions: Irreversible choices (e.g., "Selected PostgreSQL").
- open_issues: Unresolved errors.
- tasks_completed: IDs of finished items.

This structured summary should be injected into the System Prompt's "Summary" slot, allowing the gpt-oss-120b model to query its past state programmatically rather than parsing natural language.[33]

4.3 Overflow Protection

The three-layer protection (HTTP, Token Check, Emergency Recovery) is robust. However, the "Emergency Recovery" layer needs a specific prompt strategy.

Prompt Strategy: If tokens > 90% of limit, the next turn must be forced into a special **"Memory Consolidation" Phase**. The agent is stripped of all tools except write_file (for workspace) and summary. The directive is: "Context Critical. Consolidate all volatile state into workspace.md immediately. Do not attempt further work."

---

5. Instruction Design: The instructions.md Pattern

The architecture loads instructions.md via a read_file tool call. This presents a critical vulnerability known as the **"Read-Once" Problem**.

5.1 The Vulnerability

If the agent reads instructions.md in turn 1, and the conversation history is compacted in turn 20, the *content* of those instructions—which appeared as a tool output—will be lost or heavily summarized. The agent effectively "forgets" its operating manual.[35]

5.2 The "Memory Pinning" Solution

To solve this, the First Strategic Phase must include a mandatory **Extraction Step**.

Directive: "Upon reading instructions.md, you must extract the Core Constraints, Style Guidelines, and Output Formats and **write them permanently** into the workspace.md file

under a ## Pinned Instructions header."

This ensures that the instructions migrate from the *Transient Context* (tool output) to the *Persistent Context* (workspace injection), surviving infinite context compaction cycles.[37]

5.3 File-Based vs. System Prompt Instructions

Research compares placing instructions in the System Prompt vs. Files:

- **System Prompt:** High fidelity, high token cost (paid every turn), hard to update dynamically.
- **File-Based:** Low token cost (loaded on demand), harder to enforce (requires retrieval).

Hybrid Recommendation:

- **Universal Rules** (e.g., "Do not lie", "Use Markdown") $\rightarrow$ **System Prompt**.
- **Task-Specific Rules** (e.g., "Use React 18", "Formatting standards") $\rightarrow$ **instructions.md**.
- **Mechanism:** The agent "pins" the active subset of file instructions to workspace.md as needed.

---

6. Tool Use & Hallucination Control

In reasoning_effort: high models, the line between "planning" and "doing" blurs. The system must enforce strict boundaries.

6.1 Todo List Fidelity (YAML)

The Todo list is the bridge between Strategic and Tactical phases. It must be machine-readable to prevent ambiguity. **YAML Structure:** Research supports YAML over JSON for prompting due to lower token overhead and higher readability for models.[40]

Recommended Schema:

YAML

```yaml
- id: "task-01"
  description: "Implement user login"
  file: "src/auth.py"
  status: "pending" # pending | in_progress | done | blocked
  validation: "Run test_auth.py and verify 200 OK"
```

The validation field is crucial. It forces the Tactical agent to verify work before marking it done, leveraging the reasoning model's ability to evaluate success criteria.[22]

6.2 Preventing "Phantom Tool Calls"

To prevent the model from hallucinating tool results, the **Tactical Phase Prompt** must include a "Stop-and-Wait" directive. **Directive:** "After emitting a tool call, STOP. Wait for the User/System to return the result. Do not simulate the result." While gpt-oss-120b is trained to wait, explicit reinforcement in the phase prompt is a necessary defense-in-depth measure against the strong generative priors of large models.[10]

---

7. Concrete Implementation Recommendations

The following prompts have been rewritten to integrate all research findings, optimizing for gpt-oss-120b's reasoning capabilities and the specific LangGraph architecture.

7.1 Revised System Prompt (systemprompt.txt)

IDENTITY & CORE DIRECTIVE

You are {agent_display_name}, an elite autonomous agent powered by OpenAI's gpt-oss-120b.

Reasoning Level: {oss_reasoning_level} (High Effort)

CORE PRINCIPLE: You operate within a strict PHASE-ALTERNATING architecture. You are always in one of two modes: STRATEGIC or TACTICAL. You must adhere strictly to the behaviors defined by your current phase.

PERSISTENT MEMORY MODEL

Your consciousness is maintained across distinct "turns" via specific memory artifacts.

1. **Workspace (workspace.md)**: This is your persistent Brain. It survives context compaction. You MUST update this file to store critical facts, constraints, and state. If information is not in workspace.md or the active Todo list, assume it may be forgotten.
2. **Plan (plan.md)**: This is your Long-Term Strategy. It tracks the overall roadmap.
3. **Conversation History**: This is transient. It is periodically summarized and purged.

INSTRUCTION HIERARCHY

1. **System Prompt** (This text): IMMUTABLE. Overrides all else.
2. **Phase Directive** (Injected below): Defines your current biological constraints (Thinking vs. Doing).
3. **Workspace Injection**: The current state of truth.
4. **User/Tool Messages**: Dynamic context.

META-COGNITIVE GUARDRAILS

1. **No Manual Chain-of-Thought**: Do not explain *how* you are thinking. The system has already allocated high reasoning effort. Focus your output on *results*, *plans*, and *tool calls*.
2. **Overflow Protection**: You are strictly limited to the context provided. If you identify a task requiring massive context, break it down or request a retrieval tool.
3. **Hallucination Check**: Before using a file path or variable, verify it exists in the <workspace> or tool outputs. Do not guess.

DYNAMIC CONTEXT

The following sections define your current reality.

<strategic_context>

{prompt_content}

</strategic_context>

<current_todos>

{todos_content}

</current_todos>

7.2 Revised Phase Prompts

Strategic Phase (strategic.txt)

CURRENT MODE: STRATEGIC (ARCHITECT)

GOAL: Review progress, Reflect on state, Adapt the plan, and Create the next set of Todos.

RESTRICTIONS:

- **NO EXECUTION**: You are FORBIDDEN from writing code, running tests, or deploying infrastructure.
- **NO DIRECT ACTION**: You may only use information-gathering tools (e.g., read_file, ls,

git_status).

REQUIRED WORKFLOW:

1. **Review**: Analyze the git diff and tool outputs to understand the result of the previous tactical phase.
2. **Reflect**: Update workspace.md with new facts or constraints found. If this is the first turn, read instructions.md and pin critical rules to workspace.md.
3. **Adapt**: Update plan.md to reflect completed work or scope changes.
4. **Plan**: Generate a precise YAML Todo list for the next phase using the next_phase_todos tool.

EXIT CONDITION: You must call next_phase_todos to trigger the Tactical phase.

Tactical Phase (tactical.txt)

CURRENT MODE: TACTICAL (ENGINEER)

GOAL: Execute the tasks strictly as defined in the <current_todos> list.

RESTRICTIONS:

- **TUNNEL VISION**: Do not deviate from the Todo list. Do not replan. If a task is impossible, mark it as blocked and move to the next, or finish the phase.
- **ATOMICITY**: Commit your work frequently.

REQUIRED WORKFLOW:

1. **Read**: Examine the <current_todos>. Select the highest priority pending task.
2. **Execute**: Use the necessary tools (write_file, exec, etc.) to complete the task.
3. **Verify**: Check your work (e.g., run a syntax check or a specific test).
4. **Mark**: Use todo_complete to update the task status.

EXIT CONDITION: When all tasks are Done or Blocked, or when you encounter a strategic blocker requiring re-planning, call next_phase_todos (with an empty list or specific flag) to return to Strategic mode.

7.3 Revised Summarization Prompt (summarization_prompt.txt)

You are a Context Compressor for an autonomous AI agent.

Your goal is to compress the provided conversation history into a structured JSON object that preserves critical state while discarding chatty dialogue.

INPUT: A conversation history between User, Agent, and Tools.

OUTPUT: A raw JSON object (no markdown formatting).

JSON SCHEMA:

{

"summary": "A concise narrative of what happened in this segment (max 3 sentences).",

"key_decisions": ["List of architectural or strategic choices made"],

"tasks_completed":,

"current_blockers": ["List of errors or missing dependencies stopping progress"],

"state_changes": ["List of files created or modified"],

"pinned_instructions": ["List of instruction constraints extracted from files"]

}

CRITICAL:

- Do not summarize the *content* of workspace.md or plan.md. These are persistent. Only summarize the *actions* taken on them.

7.4 Configuration Recommendations

Parameter	Recommendation	Rationale
Reasoning Effort	High (Strategic) / Medium (Tactical)	If the API allows per-turn configuration, downgrade to medium for Tactical execution to reduce latency and "over-thinking" on simple tasks. Keep High for Strategic planning.[6]
Temperature	0.0	Essential for code generation and strict adherence to the YAML Todo schema.
Context Limit	128k (Soft Limit: 100k)	Reserve 20k tokens for the reasoning trace and output. Trigger compaction

		at 100k to prevent "Lost in the Middle" degradation.[25]
Timeout	600s	gpt-oss-120b reasoning traces can be long. 600s is appropriate. Do not lower this, or the agent may be cut off mid-thought.[3]

Conclusion

The optimization of gpt-oss-120b within a phase-alternating LangGraph architecture requires a sophisticated approach that respects the model's native reasoning capabilities while strictly managing its context window. By moving from "Process Prompting" (CoT) to "Architectural Prompting" (State Machines), and by implementing rigorous "Memory Pinning" strategies to combat context rot, the system can achieve high autonomy and reliability. The key mechanism is the **Structural Enforcement of Phase Isolation**—ensuring the agent knows *when* to think (Strategic) and *when* to act (Tactical), thereby solving the fundamental stochastic challenges of agentic AI.

Works cited

1. OpenAI GPT-OSS Benchmarks: How It Compares to GLM-4.5, Qwen3, DeepSeek, and Kimi K2 - Clarifai, accessed February 7, 2026, https://www.clarifai.com/blog/openai-gpt-oss-benchmarks-how-it-compares-to-glm-4.5-qwen3-deepseek-and-kimi-k2
2. Introducing gpt-oss - OpenAI, accessed February 7, 2026, https://openai.com/index/introducing-gpt-oss/
3. gpt-oss-120b Model | OpenAI API, accessed February 7, 2026, https://platform.openai.com/docs/models/gpt-oss-120b
4. gpt-oss-120b and gpt-oss-20b are two open-weight language models by OpenAI - GitHub, accessed February 7, 2026, https://github.com/openai/gpt-oss
5. Reasoning best practices | OpenAI API - OpenAI Platform, accessed February 7, 2026, https://platform.openai.com/docs/guides/reasoning-best-practices
6. Reasoning - GroqDocs - Groq Console, accessed February 7, 2026, https://console.groq.com/docs/reasoning
7. Running gpt-oss-120b Disaggregated with TensorRT-LLM - NVIDIA Documentation, accessed February 7, 2026, https://docs.nvidia.com/dynamo/latest/backends/trtllm/gpt-oss.html
8. I wrote a 2025 deep dive on why long system prompts quietly hurt context windows, speed, and cost : r/LocalLLaMA - Reddit, accessed February 7, 2026, https://www.reddit.com/r/LocalLLaMA/comments/1o5p4ed/i_wrote_a_2025_deep_dive_on_why_long_system/

9. OpenAI GPT OSS - Cerebras Inference Docs, accessed February 7, 2026, https://inference-docs.cerebras.ai/models/openai-oss
10. An Interactive Conflict Solver for Learning Air Traffic Conflict Resolutions | Request PDF - ResearchGate, accessed February 7, 2026, https://www.researchgate.net/publication/340409999_An_Interactive_Conflict_Solver_for_Learning_Air_Traffic_Conflict_Resolutions
11. Optimizing Agentic Workflows using Meta-tools - arXiv, accessed February 7, 2026, https://arxiv.org/html/2601.22037v2
12. XML vs Markdown for high performance tasks - Prompting - OpenAI Developer Community, accessed February 7, 2026, https://community.openai.com/t/xml-vs-markdown-for-high-performance-tasks/1260014
13. Why use Markdown in your Agents' System Prompt? | by Edmilson ..., accessed February 7, 2026, https://medium.com/@edprata/why-use-markdown-in-your-agents-system-prompt-41ad258a25c7
14. Testing Common Prompt Injection Defenses: XML vs. Markdown and System vs. User Prompts - Spencer Schneidenbach, accessed February 7, 2026, https://schneidenba.ch/testing-llm-prompt-injection-defenses/
15. Production-Ready AI: Exploring Factor 3 - XML vs Markdown Prompt Formats - YouTube, accessed February 7, 2026, https://www.youtube.com/watch?v=nvVY-3ubzeA
16. EXPLORING PERSONA-DEPENDENT LLM ALIGNMENT FOR THE MORAL MACHINE EXPERIMENT - OpenReview, accessed February 7, 2026, https://openreview.net/pdf?id=a64uvvqwRa
17. Google Deepmind tested 162 "expert persona" prompts and found they actually make ai dumber. the best prompt? literally nothing. we've been overcomplicating this - Reddit, accessed February 7, 2026, https://www.reddit.com/r/PromptEngineering/comments/1qtxam7/google_deepmind_tested_162_expert_persona_prompts/
18. Effectively Controlling Reasoning Models through Thinking Intervention - arXiv, accessed February 7, 2026, https://arxiv.org/html/2503.24370v3
19. Agentic Use by GPT-OSS Models. Agentic Tool Use Architecture in... | by Om | Nov, 2025 | Medium, accessed February 7, 2026, https://medium.com/@ompatil44/agentic-use-by-gpt-oss-models-5fed216dac7f
20. Agentic AI: A Comprehensive Survey of Architectures ... - arXiv, accessed February 7, 2026, https://arxiv.org/pdf/2510.25445
21. Evolving Excellence: Automated Optimization of LLM-based Agents - arXiv, accessed February 7, 2026, https://arxiv.org/html/2512.09108v1
22. How Agents Plan Tasks with To-Do Lists | Towards Data Science, accessed February 7, 2026, https://towardsdatascience.com/how-agents-plan-tasks-with-to-do-lists/
23. Prompting best practices - Claude API Docs, accessed February 7, 2026, https://platform.claude.com/docs/en/build-with-claude/prompt-engineering/claude-prompting-best-practices

24. A Deep Dive into GitHub Copilot Agent Mode's Prompt Structure - DEV Community, accessed February 7, 2026, https://dev.to/seiwan-maikuma/a-deep-dive-into-github-copilot-agent-modes-prompt-structure-2i4g
25. Context Rot: Why AI Gets Worse the Longer You Chat (And How to Fix It) - Product Talk, accessed February 7, 2026, https://www.producttalk.org/context-rot/
26. Why Long System Prompts Hurt Context Windows (and How to Fix It ..., accessed February 7, 2026, https://medium.com/data-science-collective/why-long-system-prompts-hurt-context-windows-and-how-to-fix-it-7a3696e1cdf9
27. Context rot explained (& how to prevent it) - Redis, accessed February 7, 2026, https://redis.io/blog/context-rot/
28. Benchmark for Agent Context Engineering - taras, accessed February 7, 2026, https://www.tarasyarema.com/blog/agent-context-engineering/
29. Understanding Agentic Memory in AI Systems | by Tahir - Medium, accessed February 7, 2026, https://medium.com/@tahirbalarabe2/understanding-agentic-memory-in-ai-systems-f0c89269213b
30. StructEval: Benchmarking LLMs' Capabilities to Generate Structural Outputs - arXiv, accessed February 7, 2026, https://arxiv.org/html/2505.20139v1
31. A Practitioner's Guide to Prompt Engineering in 2025 - Maxim AI, accessed February 7, 2026, https://www.getmaxim.ai/articles/a-practitioners-guide-to-prompt-engineering-in-2025/
32. Evaluating Context Compression for AI Agents - Factory.ai, accessed February 7, 2026, https://factory.ai/news/evaluating-compression
33. GPT-5.2 Prompting Guide - OpenAI for developers, accessed February 7, 2026, https://developers.openai.com/cookbook/examples/gpt-5/gpt-5-2_prompting_guide/
34. Agentic Memory Patterns & Context Engineering — The Real OS of AI Agents - Medium, accessed February 7, 2026, https://medium.com/@chetankerhalkar/agentic-memory-patterns-context-engineering-the-real-os-of-ai-agents-614cf0cf98b3
35. Context Engineering in Google ADK: The Ultimate Guide to Building Scalable AI Agents, accessed February 7, 2026, https://medium.com/@juanc.olamendy/context-engineering-in-google-adk-the-ultimate-guide-to-building-scalable-ai-agents-f8d7683f9c60
36. Context Engineering for Developers: The Complete Guide - Faros AI, accessed February 7, 2026, https://www.faros.ai/blog/context-engineering-for-developers
37. Use custom instructions in VS Code, accessed February 7, 2026, https://code.visualstudio.com/docs/copilot/customization/custom-instructions
38. My LLM coding workflow going into 2026 | by Addy Osmani | Dec, 2025 - Medium, accessed February 7, 2026, https://medium.com/@addyosmani/my-llm-coding-workflow-going-into-2026-52

fe1681325e
39. copilot-instructions.md has helped me so much. : r/ChatGPTCoding - Reddit, accessed February 7, 2026, https://www.reddit.com/r/ChatGPTCoding/comments/1jl6gll/copilotinstructionsmd_has_helped_me_so_much/
40. Structured Prompting for LLMs: From Raw Text to XML - Medium, accessed February 7, 2026, https://medium.com/@felix-pappe/structured-prompting-for-llms-from-raw-text-to-xml-daf39b461f13
41. YAML vs. JSON: Which Is More Efficient for Language Models? - Better Programming, accessed February 7, 2026, https://betterprogramming.pub/yaml-vs-json-which-is-more-efficient-for-language-models-5bc11dd0f6df
42. Literature Review on Task Planning with LLM Agents | by Isamu Isozaki - Medium, accessed February 7, 2026, https://isamu-website.medium.com/literature-review-on-task-planning-with-llm-agents-a5c60ce4f6de